MASTER EN SCIENCES INFORMATIQUES

Systèmes d'autorisations : différents modèles pour des approches différentes

Claudy, Thierry

Award date:
1988

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES N.-D. DE LA PAIX - NAMUR

Institut d'Informatique

SYSTEMES D'AUTORISATIONS : Différents modèles pour des approches différentes

CLAUDY Thierry

Promoteur : Jean RAMAEKERS

Mémoire présenté
en vue de l'obtention
du titre de
Licencié et Maître
en Informatique

ANNEE ACADEMIQUE 1987 - 1988

Systèmes d'autorisations : différents modèles pour des approches différentes.

Dans un contexte d'expansion du monde informatique et des risques qu'il introduit ou accentue, le présent mémoire a pour but de présenter un type d'outil diminuant ces risques (système d'autorisations) et d'en évaluer un cas bien particulier.

Ce travail souligne l'existence de deux approches totalement antagonistes dans l'étude de tels systèmes, la première visant à construire des modèles théoriques en vue d'une étude approfondie de ce domaine et la seconde visant plutôt à construire des modèles pratiques permettant de construire des systèmes d'autorisations opérationnels. Pour chacune de ces approches, nous étudions un modèle particulier et nous montrons quel peut être le lien entre deux modèles issus d'approches diamétralement opposées. Nous terminons par la construction d'un système d'autorisations issu du second modèle (approche pratique) afin d'en évaluer les caractéristiques.

CLAUDY Thierry
2ème Lic. et Maîtrise
Informatique.

Authorization systems : various models for various approaches.

In a context of expansion of data processing and of the risks it introduces or increases, the aim of this thesis is to present a kind of tool which is able to decrease these risks (authorization systems) and to evaluate a particular instance of such a type.

This work underlines the existence of two approaches that are poles apart in the study of such systems, the former aiming at producing theoretical models useful for a thorough study of this domain, and the latter aiming at producing practical models allowing to build operational authorization systems. In order to illustrate each method, we study for each a particular model and we show the link between these two models proceeding from quite opposed approaches. We conclude with the building of an authorization system arising from the second model (practical approach) so as to evaluate its features.

CLAUDY Thierry
2ème Lic. et Maîtrise
Informatique.

R E M E R C I E M E N T S

Nous tenons à remercier Monsieur RAMAEKERS, Mademoiselle MAHIAT et Monsieur BOUXKHA pour l'aimable attention et les conseils qu'ils ont portés à la réalisation de ce mémoire.

Nous tenons également à remercier Monsieur WEHENCKEL de l'Institut Supérieur de Technologie à Luxembourg pour l'autorisation donnée à l'utilisation de leur machine.

TABLE DES MATIERES

AVANT - PROPOS

Avant toute chose, il est utile d'avertir le lecteur qu'il dispose d'une liste récapitulative des abréviations utilisées dans ce mémoire; cette liste se trouve en annexe B à la page 138 du présent tome.

Il existe, d'autre part, un second tome traitant de la partie pratique de ce mémoire et aboutissant à un logiciel dont le code source est disponible chez Monsieur Ramaekers.

I N T R O D U C T I O N

La place sans cesse croissante qu'occupe l'informatique dans les entreprises modernes a pour conséquence une dépendance de plus en plus marquée de ces entreprises vis-à-vis du bon fonctionnement des outils informatiques et de leur utilisation à bonnes fins.

Il est dès lors devenu vital pour une entreprise de pouvoir évaluer les risques informatiques qu'elle encourt et de pouvoir diminuer ces risques autant qu'économiquement rentable. Il y a là, en effet, un compromis économique à faire entre investissements à des fins de diminution des risques informatiques et gains escomptés de par cette diminution.

Le présent mémoire a pour but, dans ce contexte, de présenter un type d'outil diminuant les risques informatiques (système d'autorisations) et d'en évaluer d'un point de vue technique, un cas bien particulier.

Le premier chapitre présente le type d'outil qu'est le système d'autorisations en répondant aux questions : "Qu'est-ce qu'un système d'autorisations ?", "A quoi s'applique-t-il ?", "Quels sont les risques qu'il peut réduire ?", "Quelles sont les approches possibles dans l'étude de tels systèmes?".

Le second chapitre présente un modèle de systèmes d'autorisations découlant d'une première approche dans l'étude des systèmes d'autorisations; ce modèle est dit général au sens où tout système d'autorisations couramment utilisé actuellement peut être retrouvé par instanciation de ce modèle.

Le troisième chapitre présente un modèle conceptuel (ACTEN) permettant de construire des systèmes d'autorisations, illustre la généralité du modèle du second chapitre en montrant comment, partant de ce modèle général, on peut retrouver la famille des systèmes d'autorisations issue de ACTEN, étudie un cas concret d'utilisation d'ACTEN (protection des ressources d'un centre hospitalier) et montre que ce modèle découle d'une seconde approche dans l'étude des systèmes d'autorisations.

Le quatrième chapitre apporte des extensions et modifications au modèle ACTEN en vue de construire un système d'autorisations utilisable en pratique; il souligne donc certaines insuffisances du modèle conceptuel.

Le cinquième chapitre discute de la construction d'un système d'autorisations issu du modèle ACTEN étendu au quatrième chapitre; il donne les spécifications du système d'autorisations que l'on veut construire ainsi qu'une solution conceptuelle (le passage de la solution conceptuelle à la solution physique est décrit dans le second tome).

Le sixième et dernier chapitre décrit les outils et la méthode utilisés pour effectuer les mesures de performances sur le système construit et donne les résultats et conclusions de ces mesures.

C H A P I T R E I

SYSTEMES D'AUTORISATIONS : PRESENTATION GENERALE

1.1. INTRODUCTION

L'informatique se caractérise par la concentration d'informations et de procédures codées de traitement de ces informations.

Le centre de traitement informatique a pour fonction de conserver et d'exploiter des données et des procédures pour d'autres services de la même organisation ou pour des clients extérieurs. Il doit dès lors garantir à ses clients une protection efficace des informations qui lui sont confiées. Cette protection doit être considérée sous trois aspects :

- Garantie de secret : assure au client que les données considérées comme confidentielles ne risquent pas d'être divulgées à des personnes indésirables.

- Garantie d'intégrité : assure au client que les données exploitées sont bien celles qu'il croit, qu'elles se trouvent dans un état cohérent avec l'image que le client se fait de ces données, et qu'elles sont exploitées selon les procédures qu'il a définies

- Contrainte de disponibilité : assure au client que les traitements peuvent avoir lieu avec les moyens prévus dans les délais prévus.

Le problème de protection se complique par le fait d'un des aspects clés du traitement informatique moderne qu'est le partage des ressources et des données entre plusieurs utilisateurs. Cet aspect surgit du concept de division du travail.

Il apparaît dès lors essentiel de mettre en place un système assurant le secret, limitant les traitements et les personnes pouvant effectuer ces traitements.

1.2. QU'EST-CE QU'UN SYSTEME D'AUTORISATIONS ?

Commençons tout d'abord par introduire quelques définitions qui nous seront utiles dans la suite de la discussion :

Possesseur d'une information : On appelle "possesseur d'une information" la personne responsable de l'information vis-à-vis du client et vue par le système informatique comme le propriétaire de l'information.

Sujet : on appelle "sujet" toute entité interne au système informatique pouvant avoir un rôle actif de traitement d'informations (un sujet peut être l'image dans le système d'un utilisateur ou un programme du système informatique).

Le concept d'autorisations est central à tout acte de manipulation de données caractérisées par un besoin de secret et/ou d'intégrité.

Un système d'autorisations est, comme son nom l'indique, un outil hardware/software permettant à un sujet d'en autoriser d'autres à accéder à des ressources ou données informatiques. Qui dit autoriser, dit également interdire, si bien qu'un système d'autorisations permet également de restreindre l'accès aux ressources et informations. Pour ce faire, un tel système effectue un contrôle de toute demande d'accès à une ressource ou donnée en vue d'empêcher les accès non autorisés.

Etant donné le concept de partage de ressources et de données sous-jacent à tout traitement informatique moderne, il apparaît essentiel qu'un système d'autorisations agisse de manière sélective du point de vue des contrôles d'accès; en effet, dans un tel contexte, certains sujets du système informatique tels que les utilisateurs participant au projet P, devront avoir le droit d'exécuter tel programme Prg, alors que d'autres sujets ne pourront pas avoir ce droit sur ce programme.

Le concept d'autorisations sélectives sous-entend une identification fiable des sujets du système informatique. Le problème de cette identification se complique pour une catégorie spéciale de sujets, celle des

utilisateurs du système informatique: en effet, pour cette catégorie de sujets, s'ajoute un problème de correspondance entre l'ensemble des sujets de cette catégorie et l'ensemble des personnes physiques représentées par ces sujets; cette correspondance doit être telle que tout élément de l'ensemble des sujets-utilisateurs sera toujours associé à la même personne physique; nous introduisons cette propriété en tant qu'hypothèse préalable à l'utilisation d'un système d'autorisations bien qu'en toute généralité on ne puisse pas affirmer que cette propriété se vérifie dans le cadre d'un système informatique.

Un système d'autorisations a pour but de contrôler et de restreindre les accès aux ressources et données d'un système informatique tout en étant transparent au maximum afin de ne pas trop gêner les utilisateurs. Pour ce faire, il se constitue une "liste d'autorisations" sur base des autorisations que tout possesseur d'information veut bien donner à d'autres sujets.

Un modèle de système d'autorisations est une structure représentant un système d'autorisations général au sens où il représente une famille de systèmes d'autorisations et où une instanciation particulière des éléments de cette structure permet d'obtenir un système d'autorisations particulier de cette famille.

Un modèle conceptuel de systèmes d'autorisations est une structure d'aide à la conception des systèmes d'autorisations. Il doit permettre de spécifier quelles sont les entités à protéger ou dont il faut restreindre les droits d'accès et quelles sont les contraintes mises sur le comportement de ces entités, de spécifier les schémas d'accès désirés, les règles régissant l'évolution de ces schémas d'accès.

1.3. HYPOTHESES PREALABLES A L'UTILISATION D'UN SYSTEME D'AUTORISATIONS.

a. Le système informatique considéré propose un mécanisme infaillible d'identification d'utilisateur en tant que personne physique.

b. Le logiciel d'exploitation du système informatique considéré est sans erreur et offre la sécurité dans son utilisation.

c. Tout accès à une ressource ou donnée du système informatique aura nécessairement été précédé par une demande d'accès au système d'autorisations que l'on mettra en place.

1.4. QUELLE EST LA PROTECTION OFFERTE PAR UN SYSTEME D'AUTORISATIONS ?

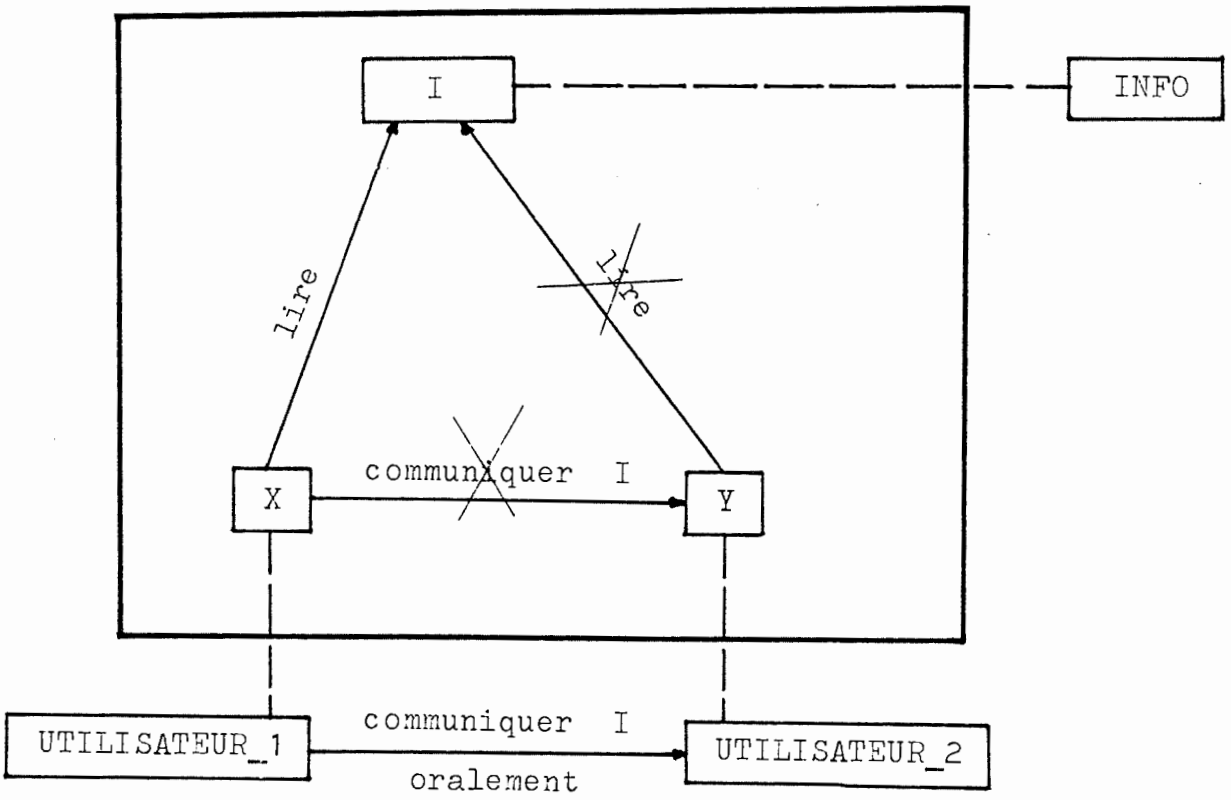
Nous avons vu au paragraphe 1.1. que la protection qu'un centre informatique devrait garantir à tout propriétaire d'informations utilisant ses services portait sur trois points :

- le secret;
- l'intégrité;
- la disponibilité.

Un système d'autorisations contribue, moyennant les hypothèses posées au paragraphe 1.3., à assurer cette protection. Dans quelle mesure un tel système offre-t-il ces trois points de protection ?

Garantie de secret : un système d'autorisations ne peut assurer qu'une information ne peut être divulguée; la raison en est qu'il faut ici distinguer l'information proprement dite de sa représentation codée à l'intérieur d'un système informatique car il existe un circuit de divulgation d'information extérieur même à un tel système (circuit de

divulgation orale) qui implique, comme le montre la figure 1-1, que même si un utilisateur n'a pas le droit de prise de connaissance d'une information codée dans le système informatique, il lui est possible d'obtenir cette information par voie orale c'est-à-dire qu'un autre utilisateur ayant un droit de prise de connaissance de la représentation codée peut communiquer cette information via le circuit oral extérieur s'il ne peut utiliser un circuit intérieur au système informatique.



X = représentation que se fait le système de l' UTILISATEUR_1
Y = représentation que se fait le système de l' UTILISATEUR_2
I = représentation que se fait le système de l'information INFO

FIGURE 1-1. Flux extérieur d'information

Il est dès lors clair qu'un système d'autorisations ne peut assurer la confidentialité d'une information que via les circuits de flux d'informations intérieurs à un système informatique.

Garantie d'intégrité : un système d'autorisations peut garantir au possesseur d'une information (et non au client) que les données exploitées sont bien celles qu'il croit, qu'elles sont exploitées selon les procédures

qu'il a définies et que les données se trouvent dans un état cohérent avec l'image que le possesseur se fait de ces données. Ainsi, en ajoutant l'hypothèse que le comportement du possesseur reflète exactement la volonté du client, un système d'autorisations peut assurer l'intégrité des données, pour autant qu'elles ne soient pas menacées par les catastrophes naturelles.

Disponibilité : un système d'autorisations ne peut ici garantir qu'une seule chose, découlant de la garantie d'intégrité, et qui est le fait que le possesseur d'une information dispose de moyens lui permettant de s'assurer que son information sera disponible lorsqu'il voudra commencer à la traiter: un système d'autorisations ne garantit rien sur les délais de traitement d'une information.

Il apparaît qu'un système d'autorisations est nécessaire pour assurer une protection efficace des informations confiées à un centre informatique, mais n'est certainement pas suffisant, loin de là; un système d'autorisations est donc un élément particulier d'un ensemble de mesures de prévention de violation de l'intégrité, du secret et de la disponibilité d'une information confiée à un tel centre.

1.5. QU'ENTEND-ON PAR SECURITE D'UN SYSTEME D'AUTORISATIONS ?

Un responsable de la sécurité peut se poser bon nombre de questions concernant un système d'autorisations et plus particulièrement concernant la sécurité d'un système d'autorisations. Il convient tout d'abord de définir ce que l'on entend par "sécurité d'un système d'autorisations".

Une première définition de cette notion pourrait exprimer qu'un système d'autorisations offre la sécurité si et seulement si l'accès à une information est impossible pour quiconque n'est pas possesseur de cette information. Cependant, les systèmes d'autorisations doivent offrir la possibilité de déléguer des autorisations (division du travail) et permettre ainsi à des entités d'accéder à une information qu'elles ne possèdent pas.

Une deuxième définition pourrait exprimer qu'un système d'autorisations offre la sécurité si et seulement si il n'existe aucune possibilité d'acquisition de droit. Ceci signifie que d'autres sujets que

le possesseur peuvent disposer d'un droit d'accès sur une information mais que ces sujets et ces droits doivent être définis une fois pour toutes (à l'initialisation du système d'autorisations). Un système d'autorisations offrant la sécurité serait dès lors un système entièrement statique et ne conviendrait pas pour représenter une structure en perpétuelle évolution. Or, le monde réel ne peut faire apparaître que des structures en évolution car évoluer et s'adapter à son environnement est une condition essentielle à la survie de toute structure réelle.

Une troisième définition pourrait exprimer qu'un système d'autorisations offre la sécurité si et seulement si un sujet autre que le possesseur d'une information ne peut en aucun cas donner un droit d'accès sur cette information à un sujet non fiable, c'est-à-dire un sujet auquel le possesseur de l'information ne fait pas confiance. Mais qu'est-ce qu'un sujet fiable ? Que représente pour un système d'autorisations la confiance d'un sujet en un autre ? Un sujet peut-il être fiable en ce qui concerne un droit d'accès x à une information et non fiable en ce qui concerne un droit d'accès y à cette information ? Un sujet est-il fiable ou non fiable une fois pour toutes ou peut-il acquérir ou perdre dynamiquement la confiance du possesseur d'une information ? Voilà une liste de questions auxquelles il faut répondre pour compléter cette troisième définition. Différentes réponses à ces questions peuvent être envisagées chacune donnant une définition différente de la notion de sécurité d'un système d'autorisations.

Exemple de réponses complétant la troisième définition :

La fiabilité d'un sujet est fonction du droit et de l'information considérés un sujet peut donc être fiable en ce qui concerne un droit d'accès X sur une information I et en ce qui concerne un droit d'accès Y sur une information J. et non fiable en ce qui concerne un droit d'accès X sur une information J et en ce qui concerne un droit d'accès Y sur une information I. Le système d'autorisations se représente donc un sujet fiable en tant qu élément d'une liste raccrochée à un droit D, une information I.

Si l'on décide que la notion de fiabilité d'un sujet est une notion dynamique (évoluant en cours de fonctionnement du système d'autorisations),

cette liste devient dynamique et, comme le montre la figure 1-2, est non seulement raccrochée à un droit D et une information I mais encore au possesseur de l'information P qui est le seul à pouvoir modifier cette liste.

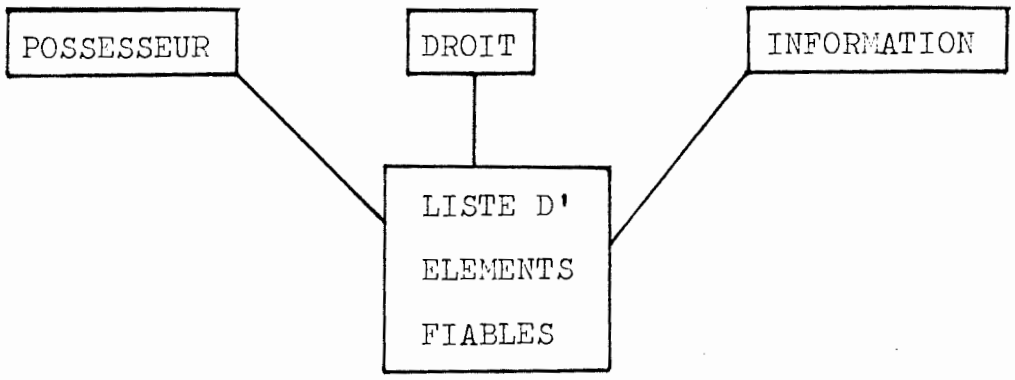


FIGURE 1-2. Liste d'éléments fiables

L'élimination d'un sujet d'une liste de fiabilité doit alors s'accompagner, pour garantir la sécurité, d'un arrêt total de l'exploitation du système informatique par les utilisateurs afin de vérifier si le sujet nouvellement éliminé n'est pas précisément en train d'effectuer l'accès auquel il n'a plus droit, et, le cas échéant, de l'empêcher de continuer cet accès. Assurer la sécurité demande dès lors un surcroît de travail à un système d'autorisations et accroît donc la gêne d'un sujet dans l'exécution de son travail (diminution de la transparence du système).

Une quatrième définition pourrait exprimer qu'un système d'autorisation offre la sécurité si et seulement si le fait qu'un sujet puisse acquérir un droit d'accès sur une information résulte directement ou indirectement d'une ou plusieurs actions entreprises par le possesseur de l'information et est connu de celui-ci. Cette dernière définition est la meilleure car n'enlève rien à la souplesse d'utilisation d'un système d'autorisations (ce qui n'est pas le cas des trois premières) et ajoute une dimension informationnelle supplémentaire à tout système assurant la sécurité ainsi définie (le possesseur d'une information sait à tout moment ce que l'on peut faire à son information).

1.6. INDECIDABILITE OU DECIDABILITE DU PROBLEME DE SECURITE D'UN MODELE DE SYSTEMES D'AUTORISATIONS.

Le problème de sécurité d'un modèle de systèmes d'autorisations est celui de déterminer si un quelconque système issu de ce modèle offre la sécurité; ce problème ne prend donc une signification qu'après avoir opté pour une définition de la sécurité d'un système d'autorisations.

Dire que le problème de sécurité d'un modèle de systèmes d'autorisations est indécidable revient à affirmer qu'il n'existe aucun moyen algorithmique permettant de déterminer si un quelconque système issu de ce modèle offre la sécurité.

1.7. LES DEUX APPROCHES ANTAGONISTES DANS L'ETUDE DES SYSTEMES D'AUTORISATIONS.

Les deux approches antagonistes dans l'étude des systèmes d'autorisations aboutissent à deux types de modèles de systèmes d'autorisations bien distincts mais néanmoins complémentaires.

La première approche consiste à se donner un modèle général de ce qu'est un système d'autorisations et d'en étudier les caractéristiques, propriétés et limites d'un point de vue théorique en utilisant des domaines tels que la théorie des langages ou la théorie de la calculabilité; cette démarche ne met en évidence que des modèles théoriques n'ayant aucune portée pratique.

La seconde approche consiste à se donner un modèle de conception de systèmes d'autorisations en vue de construire des systèmes utilisables en pratique (répondant à des besoins pratiques), ces modèles ne sont pas généraux comme le sont ceux issus de la première approche et ne servent en rien à l'étude théorique du domaine des systèmes d'autorisations.

L'avenir dans le domaine des systèmes d'autorisations réside dans la découverte, grâce à l'étude théorique, des frontières où l'on passe d'un modèle inutilisable en pratique à un modèle utilisable en pratique ou vice-versa. En effet, la découverte de telles frontières permettrait de construire des modèles de conception permettant toujours de donner naissance à des systèmes utilisables en pratique tout en présentant des schémas d'accès beaucoup plus compliqués (plus proches de ceux que présentent les modèles théoriques actuels issus de la première approche).
En d'autres termes, la découverte de ces frontières permettrait de construire des systèmes où des libertés beaucoup plus grandes seraient laissées quant à la propagation des droits tout en gardant un problème de sécurité décidable.

C H A P I T R E II

PRESENTATION D'UN MODELE GENERAL DE SYSTEME D'AUTORISATIONS

ISSU DE L'APPROCHE THEORIQUE

2.1. INTRODUCTION

Le modèle étudié dans ce chapitre est un modèle général de ce qu'est un système d'autorisations et résulte d'une approche théorique à l'étude de ce domaine. Ce modèle dont la présentation et la discussion couvriront ce chapitre, est présenté par E.L.Leiss dans Principles of Data Security (1) et est dû à Harrison, Ruzzo et Ullman (3).

2.2. PRESENTATION DU MODELE ET DE SON PROBLEME DE SECURITE.

2.2.1. LE MODELE.

Le modèle en question comprend un nombre fini de types d'entités différents; ces types d'entités sont fixés à l'avance et sont par exemple les types sujets et objet; une entité-sujet est une entité pouvant avoir (n'ayant pas nécessairement) des droits d'actions sur d'autres entités du modèle; une entité-objet est une entité pouvant supporter (ne supportant pas nécessairemement) des droits d'actions de la part d'autres entités. Ainsi, un utilisateur et un programme du modèle sont deux exemples d'entité-sujet, un programme et une zone de données du modèle sont deux exemples d'entité-objet. Il est par conséquent évident qu'une même entité peut tout à la fois être sujet et ojbet en ce sens qu'elle peut à la fois supporter et posséder des droits d'actions.

Le modèle consiste en un ensemble fini R de droits génériques ou droits de base et un ensemble fini C de commandes. La figure 2-1 donne un exemple d'ensemble R ainsi que la forme de toute commande c de C; elle permet de voir qu'une commande c comporte un ensemble de paramètres formels et est constituée de deux parties : la première est un ensemble de m tests d'existence de droits génériques de R entre deux paramètres de la commande ($m \geqslant 0$); la seconde n'est exécutée que si les m tests précédents sont vérifiés et consiste en n opérations de base ($n \geqslant 1$) constituées chacune d'une primitive agissant sur un paramètre de la commande ou sur l'ensemble des droits existant entre deux de ces paramètres.

COMMAND c (X_1 ,..., X_k)

IF r_1 in (X_{s_1} , X_{o_1})
⋮
r_m in (X_{s_m} , X_{o_m})

THEN OP_1
⋮
OP_n

où $m \geq 0$, $n \geq 1$

r_1 ,...,r_m sont des droits de R

$1 \leq s_1$, o_1 ,..., s_i , o_i ,..., s_m , $o_m \leq k$

X_1 ,..., X_k sont les paramètres formels de la commande

chaque OP_i est l'une des six primitives suivantes :

enter r into (X_{s_i} , X_{o_i})
delete r from (X_{s_i} , X_{o_i})
create subject (X_{s_i})
delete subject (X_{s_i})
create object (X_{o_i})
delete object (X_{o_i})

R = (write , read , use)

FIGURE 2-1. Exemple d'ensemble R et forme d'une commande c de C

Etant donné un système d'autorisations (R,C), nous pouvons définir ce que l'on entend par configuration ou description instantanée du système (R,C). Une configuration de (R,C) est un triplet (S,O,P) où O est l'ensemble des objets courants, S est un sous-ensemble de O et représente l'ensemble des sujets courants, et P est une matrice d'accès comportant une ligne pour tout s de S et une colonne pour tout o de O. Il est à noter que cette définition introduit le fait que tout sujet est aussi un objet. La figure 2-2 illustre tout ceci en montrant une matrice d'accès où l'entrée P (s.o)

s'ouvre sur un ensemble de droits de R que le sujet s possède sur l'objet o: il est clair que P (s,o) est un sous-ensemble de R.

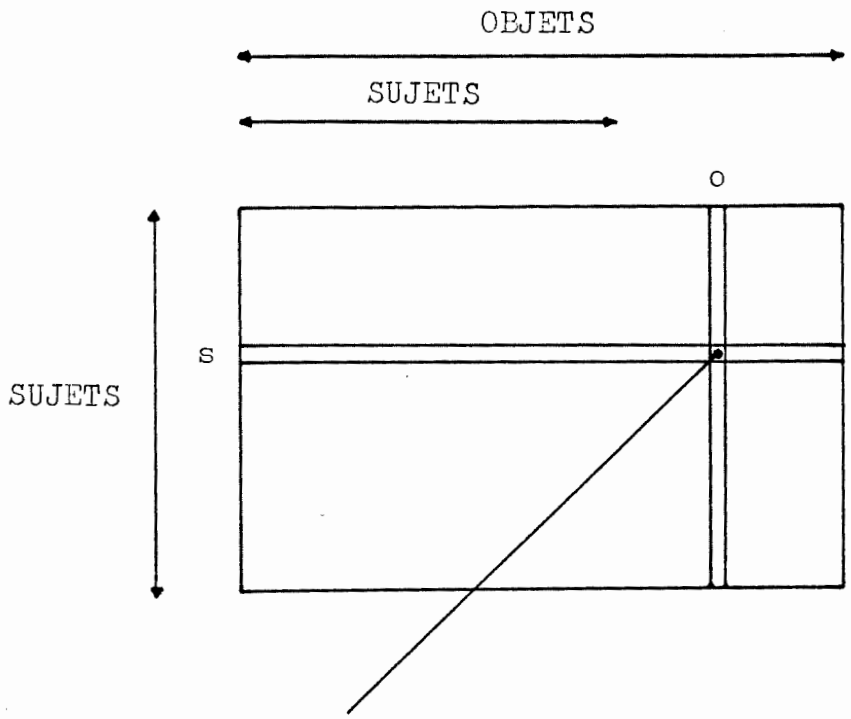


FIGURE 2-2. Matrice d'accès

Nous pouvons maintenant, à la lumière de la définition de la notion de ccnfiguration, préciser davantage l'effet des six primitives de base dont nous avons parlé lors de la présentation de la forme des commandes de C. L'effet de chacune des six primitives est défini ci-dessous par la comparaison de la configuration finale (S', O', P'), obtenue par application de la primitive à la configuration initiale (S,O,P), et de cette configuration initiale :

(1) enter r into (s,o) induit une configuration finale dont la seule différence avec la configuration initiale est l'existence d'un nouveau droit r entre s et o s'il n'existait pas dans la configuration initiale :

$S' = S,\ O' = O,\ s \in S,\ o \in O,\ s' \in S',\ o' \in O'$

$P'(s', o') = P(s', o')$ pour tout $(s', o') \neq (s,o)$

$P'(s,o) = P(s,o) \cup \{r\}$

(2) delete r from (s,o) induit une configuration finale dont la seule différence avec la configuration initiale est l'inexistence du droit r entre s et o s'il existait dans la configuration initiale :

$S' = S,\ O' = O,\ s \in S,\ o \in O,\ s' \in S',\ o' \in O'$

$P'(s',o') = P(s,o)$ pour tout $(s', o') \neq (s,o)$

$P'(s,o) = P(s,o) - \{r\}$

(3) create subject s' induit une configuration finale dont la seule différence avec la configuration initiale est l'existence d'un sujet et d'un objet supplémentaires si ce sujet n'était pas déjà repris comme objet dans la configuration initiale :

$S' = S \cup \{s'\}$, $O' = O \cup \{s'\}$ si $s' \notin O$

$P(s,o) = P(s,o)$ pour tout (s,o) de $S \times O$

$P'(s',o) = \{\}$ pour tout o de O'

$P'(s,s') = \{\}$ pour tout s de S'

(4) create object o' induit une configuration finale dont la seule différence avec la configuration initiale est l existence d'un objet supplémentaire si cet objet n'appartenait pas déjà à la configuration initiale :

$S' = S,\ O' = O \cup \{o'\}$

$P'(s,o) = P(s,o)$ pour tout (s,o) de $S \times O$

$P'(s,o') = \{\}$ pour tout s de S'

(5) delete subject s' induit une configuration finale dont la seule différence avec la configuration initiale est l'inexistence d'un sujet et objet s' si ce sujet appartenait aux sujets de la configuration initiale :

$S' = s - \{s'\}$, $O' = O - \{s'\}$ si $s' \in S$

$P'(s,o) = P(s,o)$ pour tout (s,o) de $S' \times O'$

(6) delete object O induit une configuration finale dont la seule différence avec la configuration initiale est l'inexsitence d'un objet o' si cet objet appartenait à la configuration initiale :

$S' = S$, $O' = O - \{ o' \}$, $o' \in O - S$

$P'(s.o) = P(s,o)$ pour tout (s,o) de $S' \times O'$.

Etudions maintenant comment une commande s'exécute dans ce système d'autorisations (R C). On dit qu'une configuration Q induit une autre configuration Q' sous la commande c et les paramètres réels $x_1,\ldots,x_k$, où c est

command c $(X_1,\ldots,X_k)$

if r_1 in (X_{s1},X_{o1}) and

.

.

.

r_m in (X_{s_m},X_{o_m})

then op_1

.

.

.

op_n

end

et on écrit Q -----c $(x_1,\ldots,x_k)$ Q'
si Q' est définie comme suit :

(1) si la condition de la commande c n'est pas satisfaite (il existe au moins un i de $1,\ldots,m$ tel que $r_i \notin P(x_{s_i},x_{o_i})$), alors Q' = Q:

(2) si la condition de la commande c est satisfaite alors il doit exister des configurations $Q_o,\ldots,Q_n$ telles que

$Q = Q_o$---op_1* Q_1----op_2* ...-----op_n* $Q_n = Q'$

où op_i* représente une opération primitive op_i où les paramètres formels $X_1,\ldots,X_k$ ont été remplacée par les paramètres réels $x_1,\ldots,x_k$.

Si l'on peut induire Q' à partir de Q par n répétitions de ----, $n \geq o$. on écrit Q ---* Q'.

Examinons maintenant le problème de sécurité de ce modèle.

2.2.2. Le problème de sécurité de ce modèle.

La notion de sécurité utilisée pour ce modèle général s'apparente très fort à la quatrième définition donnée au point 1.5. à la page 13 ; elle est en fait légèrement modifiée pour mieux épouser le modèle général et est moins restrictive qu'au point 1.5. On dira, en effet, qu'un système d'autorisations offre la sécurité si et seulement si il est possible de déterminer dans une configuratin quelconque s'il existe un ensemble de commandes dont l'exécution aboutirait à une configuration dans laquelle un droit quelconque existerait entre deux entités quelconques.

Le problème de sécurité de ce modèle général est donc celui de déterminer si un système d'autorisations quelconque issu de ce modèle offre la sécurité. Harrison, Ruzzo et Ullman ont montré que le problème de sécurité ainsi défini est indécidable.

2.3. Discussion du modèle.

Dans le cadre de ce modèle, l'insertion éventuelle d'un droit sur un objet dans la matrice d'accès est uniquement fonction de l'état courant de cette matrice (configuration courante) et de l'ensemble des commandes défini.

Attribuer aux entités du modèle des participations à une telle insertion n'est pas chose facile car il est clair que la notion de possesseur d'information n'existe pas directement (ce qui ne veut pas dire que cette notion ne peut apparaître par choix judicieux de l'ensemble des commandes et de l'ensemble des droits génériques).

Que le problème de sécurité soit indécidable ne signifie pas qu'il ne peut pas exister de sous-classes de systèmes d'autorisations pour lesquelles le problème de sécurité est décidable. L'obtention de sous-classes de systèmes d'autorisations résulte de la mise en place de restrictions sur ce que peut être l'ensemble des commandes. Voici quelques exemples de sous-classe de systèmes d'autorisations à problème de sécurité décidable :

Système d'autorisations mono-opérationnel : un système d'autorisations est dit mono-opérationnel si toute commande du système ne contient qu'une seule opération primitive.

E.L.Leiss montre que le problème de sécurité d'un modèle de systèmes d'autorisations mono-opérationnels est décidable.
De même en est-il du problème de sécurité d'un système d'autorisations ne comportant pas d'opération de création.

Que le problème de sécurité d'un modèle de systèmes d'autorisations soit indécidable ne signifie pas non plus qu'il ne peut y avoir d'approche heuristique pour déterminer si un système d'autorisations particulier issu d'un tel modèle offre la sécurité. On peut dès lors s'interroger sur l'utilité et l'effectivité d'un système expert pour gérer certains modèles à problème de sécurité indécidable.

Pour conclure ce chapitre et introduire le chapitre suivant, on peut dire que ce modèle général n'a qu'une portée théorique et ne sert qu'à montrer que les opérations d'un schéma d'accès réaliste sont suffisamment puissantes que pour rendre le problème de sécurité indécidable: ce modèle résulte ainsi d'une approche totalement opposée à celle consistant à rechercher des schémas suffisamment restreints que pour assurer la décidabilité du problème de sécurité. Nous étudierons dans le chapitre suivant un modèle issu d'une telle approche et nous essayerons de voir notamment comment, partant du modèle général de Harrison, Ruzzo et Ullman, on peut restreindre les schémas d'accès afin d'obtenir ce modèle restreint.

C H A P I T R E III

ETUDE D'UN MODELE DE SYSTEMES D'AUTORISATIONS ISSU DE L'APPROCHE PRATIQUE :

Le modèle ACTEN

3.1. PRESENTATION DU MODELE A C T E N.

Le modèle ACTEN, proposé par M. Fugini et G. Martella dans le périodique "Computers and Security" sous le titre "ACTEN : a conceptual model for security systems design" (2) est un modèle servant à la conception de systèmes d'autorisations.

Ce modèle illustre l'approche restrictive des schémas d'accès d'un système d'autorisations, approche qui, comme nous l'avons déjà mentionné au point 2.3., vise à offrir un problème de sécurité décidable tout en maintenant des schémas d'accès suffisants que pour permettre la délégation de pouvoirs. Le modèle ACTEN décrit et les aspects statiques d'un système d'autorisations, aspects correspondant aux modalités d'accès entre entités du système d'autorisations, et les aspects dynamiques d'un système d'autorisations, aspects correspondant à l'évolution des modalités d'accès entre entités. Ce modèle assiste la conception d'un système d'autorisations en permettant de représenter les relations de sécurité existant entre des ressources à protéger ou dont il faut restreindre les droits d'accès. Le produit de cette phase de conception est un modèle décrivant la structure d'un système d'autorisations en termes de ressources, droits d'accès et contraintes.

Le modèle ACTEN consiste d'une part en un certain nombre de concepts de base et d'autre part en une structure permettant d'intégrer ces concepts ainsi qu'en un ensemble de règles de manipulation de cette structure.

3.1.1. Les concepts de base du modèle ACTEN.

Les concepts de base du modèle ACTEN permettent de spécifier les entités à protéger ou dont il faut restreindre les droits d'accès, d'assigner des niveaux d'autorité à ces entités et de définir les états d'autorisation et de protection de ces entités.

3.1.1.1. L'élément de base : la paire ACTion - ENtité.

L'élément de base du modèle ACTEN est la paire action - entité. Une entité E_i est un composant du système d'autorisations capable d'exécuter ou de subir une action. Une action A représente une relation binaire entre entités d'un système d'autorisations et consiste en une direction, une opération de base telle que read, update, ..., ainsi que zéro, un ou plusieurs attributs appelés prédicats. La direction de l'action précise quelle est l'entité qui peut exécuter l'opération de base et quelle est l'entité qui peut la subir: on dénote la présence de la direction d'une action par deux indices, le premier référençant l'entité qui peut exécuter l'opération de base, le deuxième référençant l'entité qui peut la subir.
Les prédicats sont des conditions préalables à l'exécution d'une opération de base. Une action A_{ij} sera donc définie comme suit :

$A_{ij} = a \sim \{P_{ij}\}$
où a est une opération de base et où $\{P_{ij}\}$ est l'ensemble des conditions préalables à l'exécution de a.

Partant d'une entité E_i, on peut définir l'état d'autorisations de E_i (Sa (E_i)) comme étant l'ensemble des paires composées d'une part d'une entité E_j sur laquelle E_i possède au moins une action et d'autre part de l'ensemble des actions que E_i possède sur E_j :

$$Sa\ (E_i) = \{A_{ij}^{\ h_j} - E_j\}\ ,\ j \neq i,\ 1 \leqslant j \leqslant N,\ hj \geqslant 1$$

où N est le nombre d'entités du système.
$A_{ij}^{\ h_j}$ représente l'ensemble des h_j actions que E_i possède sur E_j;

De même, on peut définir l'état de protection de E_i, (Sp (E_i)) comme étant l'ensemble des paires composées d'une part d'une entité E_l qui possède au moins une action sur E_i et d'autre part de l'ensemble des actions que E_l possède sur E_i :

$$Sp\ (E_i) = \{A_{li}^{\ m_l} - E_l\},\ l \neq i,\ 1 \leqslant l \leqslant N,\ m_l \geqslant 1$$

où N est le nombre d'entités du système,
$A_{li}{}^{m_l}$ représente l'ensemble des m_l actions que E_l possède sur E_i.

Le nombre de valeurs différentes de j dans Sa (E_i) est le nombre n_i d'entités sur lesquelles E_i peut exécuter des actions; ces valeurs sont regroupées dans un ensemble noté $\{J\}$. Le nombre de valeurs différentes de l dans $Sp(E_i)$ est le nombre p_i d'entités pouvant exécuter des actions sur E_i; ces valeurs sont regroupées dans un ensemble noté $\{L\}$. Les cardinalités de $Sa(E_i)$ et de $Sp(E_i)$ sont respectivement :

$$|Sa(E_i)| = \sum_j h_j, \forall\, j \in \{J\}$$

$$|Sp\ (E_i)| = \sum_l m_l, \forall\, l \in \{L\}$$

3.1.1.2. Classification des entités.

Partant des états d'autorisation $Sa(E_i)$ et de protection $Sp(E_i)$ d'une entité E_i, on peut classifier cette entité en tant que :

- entité uniquement active si $Sa(E_i) \neq \{\}$ et $Sp(E_i) = \{\}$
- entité uniquement passive si $Sa(E_i) = \{\}$ et $Sp(E_i) \neq \{\}$
- entité active et passive si $Sa(E_i) \neq \{\}$ et $Sp(E_i) \neq \{\}$

3.1.1.3. Classification des opérations.

Les opérations d'un système d'autorisations peuvent se répartir en deux classes différentes : les opérations statiques et les opérations dynamiques. Une opération statique (SO) est une opération dont l'exécution ne modifie ni $Sa(E_i)$ ni $Sp(E_i)$, quelle que soit l'entité E_i considérée. Une opération dynamique (DO) est une opération dont l'exécution modifie $Sa(E_i)$ ou $Sp(E_i)$ d'au moins une entité E_i. Une action statique (SA) est une action

comportant une opération statique. Une action dynamique (DA) est une action comportant une opération dynamique. L'opération READ, dont la signification est "transférer une copie d'une entité dans la zone mémoire de l'entité qui exécute l'opération", est un exemple d'opération statique.
L'opération GRANT, dont la signification est "gratifier à une entité E_i une opération ou une action statique sur une entité E_j", est un exemple d'opération dynamique.

Les opérations du système d'autorisations que l'on veut construire sont répertoriées dans deux structures hiérarchiques différentes (une pour les opérations statiques et une pour les opérations dynamiques) où :

- chaque opération se voit assigner un niveau,
- si une entité E_i peut exécuter une opération de niveau L sous certaines conditions, elle peut également exécuter les opérations de niveau $L' \leq L$ sous les mêmes conditions.

La figure 3-1 montre une telle classification et indique que, du côté des opérations dynamiques, l'opération DELEGATE/ABROGATE a un niveau plus élevé que l'opération GRANT/REVOKE et que, du côté des opérations statiques, l'opération CREATE/DELETE a un niveau plus élevé que l'opération UPDATE qui elle-même a un niveau plus élevé que l'opération READ qui elle-même a un niveau plus élevé que l'opération USE.

DELEGATE/ABROGATE > GRANT/REVOKE

CREATE/DELETE > UPDATE > READ > USE

FIGURE 3-1. Exemple de hiérarchisation des opérations

Cette organisation hiérarchique des opérations est le mécanisme permettant la gestion sélective des droits; quand une opération de niveau L est gratifiée à une entité, toutes les opérations de niveau L' $\leq$ L le sont également, ce qui simplifie les procédures de gratification et révocation des opérations.
De plus, chaque entité se voit assigner un niveau d'autorité par le biais de la spécification du niveau maximum L d'opérations que cette entité peut effectuer dans le système d'autorisations.
Cette structure hiérarchique est donc un mécanisme permettant également de contrôler les droits indirects en ce sens que, le long d'une chaîne de droits, une entité ne peut pas gagner un droit d'accès de niveau supérieur à son niveau d'autorité.

Lorsqu'on examine une action dynamique, trois entités sont à considérer :

- l'entité qui exécute l'action dynamique,
- l'entité qui subit l'action dynamique, c'est-à-dire l'entité qui se voit gratifier de l'action statique sur laquelle porte l'action dynamique,
- l'entité sur laquelle porte l'action statique gratifiée, entité que l'on appelera entité-paramètre de l'action dynamique.

3.1.1.4. Classification des états d'autorisation et de protection.

Considérons une entité E_i ainsi que les actions dynamiques dans lesquelles E_i est impliquée.
Puisque, comme nous l'avons montré ci-dessus, une action dynamique porte sur trois entités, nous allons distinguer ici trois classes d'actions dynamiques suivant le rôle que E_i joue dans ces actions :

- Les actions dynamiques que E_i peut exécuter (A_{ij}, $j \in \{J\}$, $j \neq i$),

- Les actions dynamiques que E_i peut subir directement (A_{li}, $l \in \{L\}$, $l \neq i$),
- Les actions dynamiques que E_i peut subir indirectement (A_{km}, $k = 1,\ldots,N$, $m \in \{L\}$, $m \neq i \neq k$) en tant qu'entité-paramètre.

L'exécution d'une ou plusieurs de ces actions dynamiques peut en toute généralité, changer les cardinalités de $Sa(E_i)$ et $Sp(E_i)$.

Trois cas sont à envisager :

1. L'exécution d'une action dynamique A_{li} que E_i peut subir directement a pour effet de changer la cardinalité $|Sa(E_i)|$ et de laisser inchangées les cardinalités $|Sp(E_i)|$, $|Sa(E_l)|$ et $|Sp(E_l)|$.

2. L'exécution d'une action dynamique A_{il} par E_i a pour effet de changer la cardinalité $|Sa(E_l)|$ et de laisser inchangées les cardinalités $|Sp(E_l)|$, $|Sa(E_i)|$ et $|Sp(E_i)|$.

3. L'exécution d'une action dynamique A_{km} que E_i peut subir indirectement en tant qu'entité-paramètre a pour effet de changer les cardinalités $|Sp(E_i)|$ et $|Sa(E_m)|$ et de laisser inchangées les cardinalités $|Sa(E_i)|$, $|Sp(E_m)|$, $|Sa(E_k)|$ et $|Sp(E_k)|$.

Si toutes les actions dynamiques révoquant des droits d'actions statiques de E_i sont exécutées, $|Sa(E_i)|$ atteint sa valeur minimale· si toutes les actions dynamiques gratifiant E_i de droits d'actions sont exécutées, $|Sa(E_i)|$ atteint sa valeur maximale.

Si toutes les actions dynamiques révoquant les droits d'actions de chaque entité E_j sur E_i sont exécutées, $|Sp(E_i)|$ atteint sa valeur minimale; si toutes les actions dynamiques gratifiant des droits d'actions sur E_i à chaque entité E_j sont exécutées, $|Sp(E_i)|$ atteint sa valeur maximale.

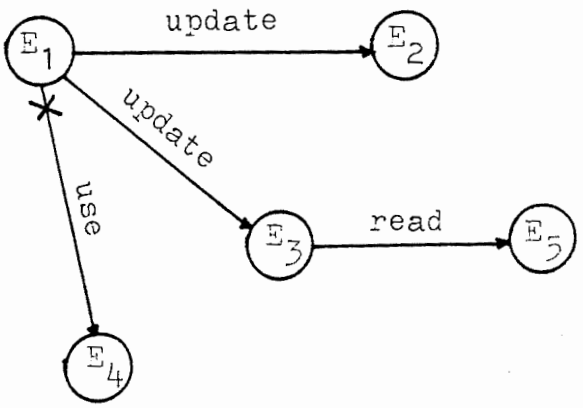
3.1.2. Structure d'intégration de ces concepts et règles de manipulation de la structure.

La structure d'intégration des concepts définis ci-dessus consiste en un graphe statique, un graphe dynamique et diverses tables.

3.1.2.1. Les graphes.

Le graphe statique SG permet de représenter les actions statiques tandis que le graphe dynamique DG permet de représenter les actions dynamiques.
L'existence de ces deux graphes bien distincts permet de séparer aspects statiques et dynamiques d'un système d'autorisations.

Le graphe statique se compose de sommets et d'arcs. Un sommet, représenté par un cercle, est associé à toute entité E_i impliquée dans au moins une action statique.
Un arc étiqueté du nom de l'opération a d'une action A représente cette action A.
L'entité origine de l'arc peut exécuter l'action A sur l'entité cible de l'arc.
Un arc agrémenté d'un symbole croisé tel que le montre la figure 3-2 indique que l'action A est conditionnée par un ensemble de prédicats $\{P\}$.
La figure 3-2 représente un exemple de SG où un utilisateur E_1 peut mettre à jour (update) les programmes d'application E_2 et E_3 sans conditions, où ce même utilisateur E_1 peut utiliser (use) le périphérique E_4 entre 10 h et 11 h du matin, et où le programme d'application E_3 peut lire (read) une zone de données E_5.



E_1 = utilisateur
E_2 = programme d'application
E_3 = programme d'application
E_4 = périphérique
E_5 = zone de données
A_{13} = use ~ "depuis 10h jusque 11h"

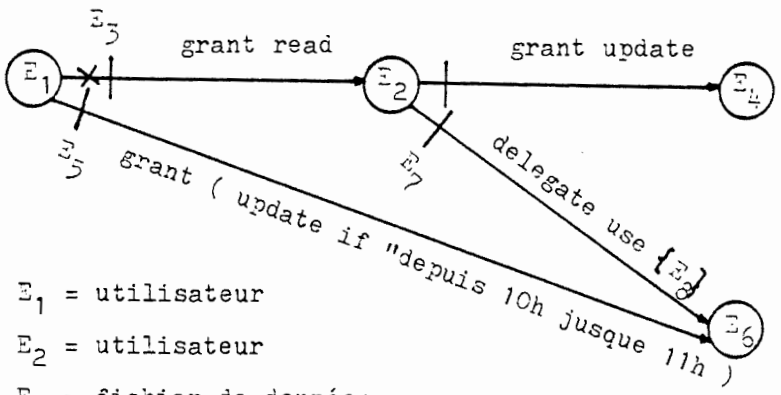
Figure 3 - 2. Exemple de graphe statique

Le graphe dynamique se compose également de sommets et d'arcs. Un sommet, représenté par un cercle, est associé à toute entité Ei impliquée directement dans au moins une action dynamique. L'adverbe "directement" permet d'exclure de la représentation du DG une entité qui ne serait impliquée dans l'ensemble des actions dynamiques qu'en tant qu'entité-paramètre. Un arc étiqueté du nom de l'opération dynamique op vise à représenter l'action dynamique comportant cette opération op. L'étiquette d'un arc représente donc le nom de l'opération associée et a donc la forme suivante :

DO SA ({ M })

où DO est l'opération dynamique,
SA est l'action statique gratifiée ou déléguée,
M est facultatif et représente, dans le cas où l'opération dynamique est "DELEGATE ou ABROGATE". l'ensemble des entités auxquelles l'entité subissant cette opération dynamique pourra éventuellement gratifier l'action statique qu'on lui délègue.

Une barre étiquetée du nom d'une entité et traversant un arc représentant une action dynamique indique l'entité-paramètre de l'action dynamique. De même que pour le graphe statique, un arc agrémenté d'un symbole croisé tel que le montre la figure 3-3 indique que l'action dynamique est conditionnée par un ensemble de prédicats { P}. Par souci de simplicité, le DG ne fait apparaître que les étiquettes GRANT ... et DELEGATE... impliquant respectivement REVOKE ... et ABROGATE... La figure 3-3 représente un exemple de DG où un utilisateur E_1 peut gratifier un utilisateur E_2 du droit de lire (read) le fichier de données E_3 sous la condition que E_2 appartienne au projet P, où l'utilisateur E_2 peut gratifier le programme d'application E_4 du droit de mettre à jour (update) le fichier de données E_5, où l'utilisateur E_2 peut déléguer au programme d'application E_6 le droit d'utiliser (use) le lecteur de cartes E_7 avec option de gratification pour le programme d'application E_8, et où l'utilisateur E_1 peut gratifier le programme d'application E_6 du droit de mettre à jour (update) le fichier de données E_5 entre 10 h et 11 h.



E_1 = utilisateur
E_2 = utilisateur
E_3 = fichier de données
E_4 = programme d'application
E_5 = fichier de données
E_6 = programme d'application
E_7 = lecteur de cartes

$A_{12/3}$ = grant read $\sim \{P\}$
$\{P\}$ = "si E_2 appartient projet P"

Figure 3-3. Exemple de graphe dynamique

3.1.2.2. Les tables.

Les tables visent à exprimer les contraintes de sécurité sur les entités et sur leurs droits.
Ces tables sont les suivantes :

- la table des classes de sécurité,
- la table des états d'entités,
- la table de hiérarchisation des opérations.

La table des classes de sécurité indique pour toute entité E_i du système d'autorisations deux contraintes :

- La potentialité active de E_i qui représente l'opération statique a_i^+ de niveau le plus élevé que E_i a le droit d'exécuter dans le système (cette potentialité active correspond donc à un niveau d'autorité),

- La potentialité passive de E_i qui représente l'opération statique a_i^- de niveau le plus élevé que E_i peut subir dans le système (cette potentialité passive correspond à la protection de E_i en ce sens que

celà permet de vérifier qu'aucun droit de niveau plus élevé que a_i ne peut être exercé sur E_i).

Cette table répond donc au besoin d'assigner à chaque entité d'un système d'autorisations un niveau d'autorité et un niveau de protection et ce, en fonction de la nature physique de l'entité, de sa position fonctionnelle dans le système et du rôle qu'elle joue dans un projet ou organisation.
La figure 3-4 représente un exemple d'une telle table où l'utilisateur E_1 peut au maximum exécuter une mise-à-jour (update) et ne peut rien subir (dû à sa nature physique), où le programme d'application E_2 peut au plus exécuter une opération de mise-à-jour (update) et peut subir au plus une opération de création/effacement (create/delete), et où le fichier de données E_4 ne peut rien exécuter et peut subir au maximum une opération de création/effacement (create/delete).

E_1	+	update
	−	———
E_2	+	update
	-	create/delete
E_3	+	———
	-	create/delete

E_1 = utilisateur
E_2 = programme d'application
E_3 = fichier de données

Figure 3-4. Table des classes de sécurité.

La table d'états d'entité spécifie quelles sont les entités possédant d'autres entités et pour chacune des entités propriétaires quelles sont les entités possédées.
La figure 3-5 est un exemple d'une telle table où l'entité E_1 possède les entités E_2, E_3 et E_4, où l'entité E_2 possède l'entité E_6 et où l'entité E_7 possède l'entité E_8.

Possesseur	Entités possédées
E_1	E_2 , E_3 , E_4
E_2	E_6
E_7	

E_1 = utilisateur

E_2 , E_3 , E_4 , E_6 = programmes d'application

E_7 = utilisateur

E_8 = fichier de données

Figure 3-5. Table d'états d'entités.

La table de hiérarchisation des opérations classe séparément les opérations statiques et dynamiques par ordre d'importance décroissante (du haut vers le bas).
La figure 3-6 représente une telle table et indique que, du côté des opérations dynamiques, l'opération DELEGATE/ABROGATE a un niveau plus élevé que l'opération GRANT/REVOKE et que, du côté des opérations statiques, l'opération CREATE/DELETE a un niveau plus élevé que l'opération UPDATE qui elle-même a un niveau plus élevé que l'opération READ qui elle-même a un niveau plus élevé que l'opération USE.

Delegate / Abrogate
Grant / Revoke

Opérations dynamiques

Create / Delete
Update
Read
Use

Opérations statiques

FIGURE 3-6. Table de hiérarchisation des opérations

3.1.3. Règles de cohérence et de transformation.

La gestion des graphes statique et dynamique s'effectue grâce aux règles suivantes :

1. Les règles assurant que chaque graphe est conforme à la réalité du système que l'on veut modéliser: ces règles, appelées règles de cohérence interne, contrôlent l'évolution des graphes statique et dynamique d'un état (configuration) en un autre.

2. Les règles assurant qu'à tout moment le graphe statique et le graphe dynamique ne se contredisent pas dans leurs descriptions d'une même réalité; ces règles ont pour nom règles de cohérence mutuelle.

3. Les règles permettant d'analyser les droits d'accès indirects et la propagation de ces droits d'accès; ces règles, appelées règles de transformation, rendent possible la détermination des états d'autorisation et de protection des entités.

3.1.3.1. Les règles du graphe statique.

Les règles de cohérence interne du graphe statique.

Dans un système d'autorisations, une entité ne doit être autorisée à exécuter et à subir que les actions compatibles avec sa nature et son rôle dans le système.
Ainsi, une exécution (use) n'a pas de sens sur un fichier de données et doit être exclue du système par les règles de cohérence interne.
Si la configuration de départ du graphe statique est cohérente avec la réalité qu'elle modélise et avec la nature physique des composants du système réel, l'évolution du graphe statique d'une configuration à une autre est régulée de façon à éviter des états inconsistants. A cette fin, l'établissement d'une nouvelle configuration entraîne l'exécution d'une routine de filtrage vérifiant la cohérence et gérant les cas incohérents.

Une telle routine examine la table de classes de sécurité afin de décider de la cohérence d'une configuration.

Les règles de transformation du graphe statique.

Les règles de transformation ont pour but d'indiquer les droits d'accès qu'une entité E_i peut exercer indirectement sur d'autres entités. Considérons une entité E_i connectée directement à une entité E_j au travers d'une action A_{ij}, supposons, comme le montre la figure 3-7, que cette entité E_j est directement connectée à une entité E_k au travers d'une action A_{jk} (on dira que A_{ij} et A_{jk} sont deux actions adjascentes). Suivant ce schéma, l'entité E_i peut exercer indirectement (via l'entité E_j) un droit d'action sur l'entité E_k.

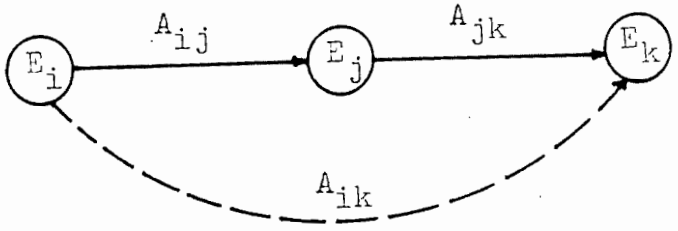


Figure 3-7. Concept de droit indirect.

La règle de transformation du graphe statique a pour but de décider quel est ce droit indirect.

La règle de transformation du graphe statique est définie sur un groupe de trois entités E_i, E_j et E_k et de deux actions adjascentes A_{ij} et A_{jk} et détermine donc l'action A_{ik} sur base de la chaîne d'action A_{ij} - A_{jk}.
Cette règle de transformation est une fonction F de quatre arguments : A_{ij}, A_{jk}, E_i et E_k.

$$A_{ik} = F\ (A_{ij}, A_{jk}, E_i, E_k)$$

Spécifications de la fonction de transformation du graphe statique.

Etant donné la table de hiérarchisation des opérations, la table des classes de sécurité, deux actions adjascentes $A_{ij} \sim a_{ij} \{P_{ij}\}$ et $A_{jk} \sim a_{jk} \{P_{jk}\}$ s'exerçant entre trois entités E_i, E_j et E_k, la fonction de transformation du graphe statique détermine l'action indirecte résultante $A_{ik} \sim a_{ik} \{P_{ik}\}$ de la manière suivante :

1. l'ensemble des prédicats $\{P_{ik}\}$ de l'action A_{ik} est la réunion de l'ensemble des prédicats $\{P_{ij}\}$ de la première action A_{ij} et de l'ensemble des prédicats $\{P_{jk}\}$ de la seconde action A_{jk}.

2. si le niveau hiérarchique de l'opération a_{ij} de la première action est supérieur ou égale au niveau hiérarchique de l'opération a_{jk} de la seconde action, alors l'opération a_{ik} de l'action indirecte résultante est aussi celle de la seconde action.

3. si le niveau hiérarchique de l'opération a_{ij} de la première action est inférieur au niveau hiérarchique de l'opération a_{jk} de la seconde action et si le niveau hiérarchique de l'opération a_{jk} de la seconde action est supérieur à celui de la potentialité active a_i^+ de E_i, alors l'opération a_{ik} de l'action indirecte résultante est la potentialité active a_i^+ de E_i.

4. si le niveau hiérarchique de l'opération a_{ij} de la première action est inférieur au niveau hiérarchique de l'opération a_{jk} de la seconde action et si le niveau hiérarchique de l'opération a_{jk} est inférieur ou égal à celui de la potentialité active a_i^+ de E_i, alors l'opération a_{ik} de l'action indirecte résultante est l'opération a_{jk} de la seconde action.

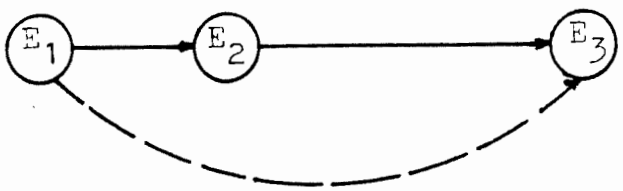
Algorithme A implémentant la fonction de transformation du graphe statique :

Si $L(a_{ij}) \geqslant L(a_{jk})$
alors $A_{ik} = a_{jk} \sim (\{p_{ij}\} \cup \{p_{jk}\})$ (1)
sinon si $L(a_{jk}) > L(a_i^+)$
alors $A_{ik} = a_i^+ \sim (\{p_{ij}\} \cup \{p_{jk}\})$ (2)
sinon $A_{ik} = a_{jk} \sim (\{p_{ij}\} \cup \{p_{jk}\})$ (3).

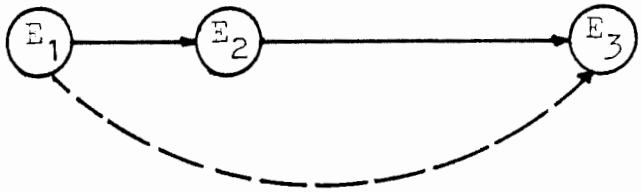
Cet algorithme ne considère pas explicitement la potentialité passive a_k^- de l'entité E_k du fait des choix d'assignation faits aux points (1), (2) et (3). Les règles de cohérence interne, assurant qu'une action sur l'entité E_k dans une configuration donnée ne peut avoir un niveau hiérarchique plus élevé que la potentialité passive a_k^- de E_k, permettent d'affirmer aux points (1) et (3) que le droit indirect résultant est de niveau hiérarchique inférieur ou égal à la potentialité passive a_k^- de E_k. D'autre part, l'assignation du point (2) permet d'affirmer également que le droit indirect résultant est de niveau hiérarchique inférieur à la potentialité passive a_k^- de E_k car $L(a_i^+) < L(a_{jk})$ et $L(a_{jk}) \leqslant L(a_k^-)$ par les règles de cohérence interne.

La figure 3-8 montre un exemple de détermination d'un droit d'action indirect: dans cette figure, un utilisateur E_1, un périphérique E_2 et un fichier de données E_3 sont représentés; la partie (a) de la figure indique le droit indirect déterminé à partir d'une première table de classes de sécurité et la partie (b) indique le droit indirect déterminé à partir d'une deuxième table de classes de sécurité.

La détermination du droit indirect A_{13} dans le cas (a) résulte de l'application du point (2) de l'algorithme A tandis que la détermination du droit indirect A_{13} dans le cas (b) résulte de l'application du point (3) de l'algorithme A.



A_{13} = update ~ ({ P_{12} } · { P_{23} })



A_{13} = create/delete ~ ({ P_{12} } · { P_{23} }

E_1	+	update
	−	———
E_2	+	create/delete
	−	use
E_3	+	———
	−	create/delete

(a)

E_1	+	create/delete
	−	———
E_2	+	create/delete
	−	use
E_3	+	———
	−	create/delete

(b)

Figure 3-8. Détermination d'un droit indirect en fonction de la potentialité active a_1^+ de E_1.

Lorsque plus d'une entité s'interpose entre E_i et E_k, le droit indirect A_{ik} est déterminé en appliquant la règle de transformation à des groupes de deux actions adjascentes et en remplaçant à chaque pas l'action déterminée au pas précédent.
Ainsi, si, comme le montre la figure 3-9, une entité E_1 est connectée à une entité E_2 par une action A_{12}, l'entité E_2 est connectée à une entité E_3 par une action A_{23} et l'entité E_3 est connectée à une entité E_4 par une action A_{34}, la détermination de l'action indirecte A_{14} se fait en deux étapes :

1. Considérer E_1, E_2, E_3, A_{12} et A_{23} et appliquer la règle de transformation pour déterminer l'action indirecte A_{13} (représenté par la partie (a) de la figure 3-9).

2. Considérer E_1, E_3, E_4, A_{13} et A_{34} et appliquer la règle de transformation pour déterminer l'action indirecte A_{14} (représenté par la partie (b) de la figure 3-9).

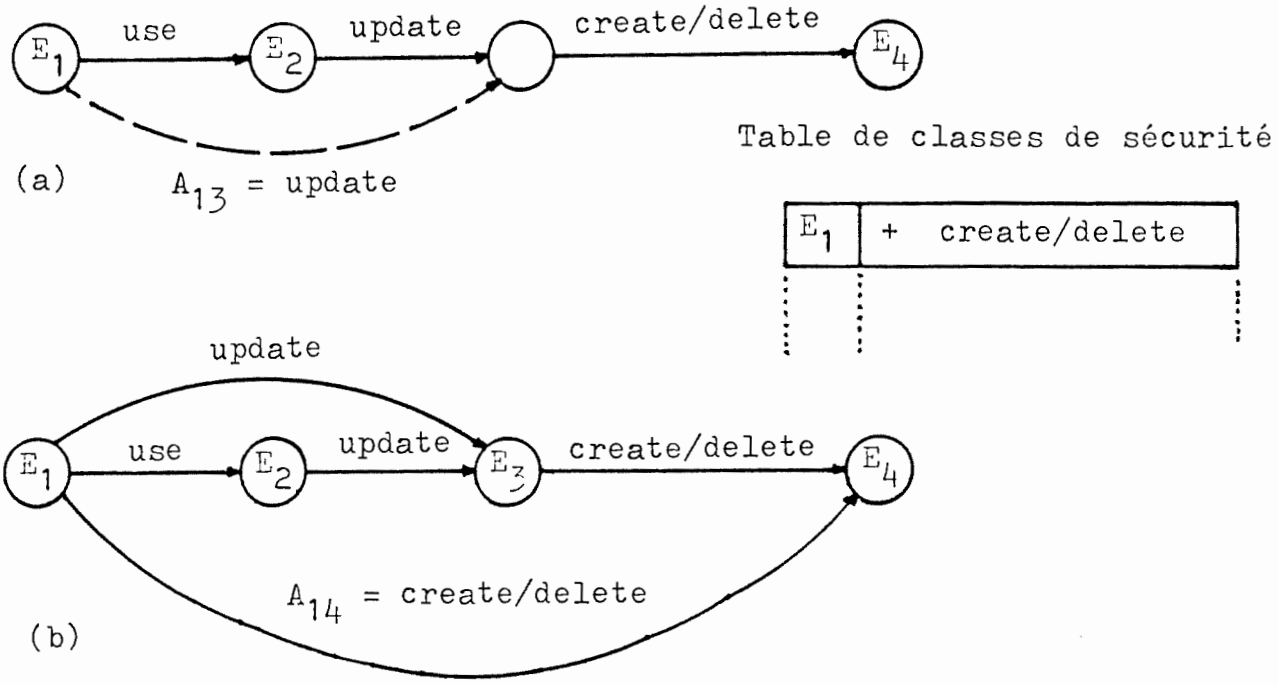


Figure 3-9. Détermination d'un droit indirect avec interposition de plus d'une entité.

3.1.3.2. Les règles du graphe dynamique.

Les règles de cohérence interne du graphe dynamique.

Règle I1. Soit E_j l'entité propriétaire d'une entité E_k. Aucune autre entité E_i ($k \neq i \neq j$) ne peut exécuter une action $A_{ij/k}$ de type GRANT/REVOKE SA où E_k est l'entité-paramètre.

En fait, si E_j est propriétaire de E_k, elle peut, par définition, exécuter une action statique SA quelconque sur E_k.
Cette règle permet d'éviter le fait qu'une entité E_i quelconque retire (revoke) à E_j les droits qu'elle possède sur l'entité E_k dont elle est propriétaire.

Règle I2. Seule l'entité E_i propriétaire d'une entité E_k peut exécuter une action $A_{ij/k}$ de type DELEGATE/ABROGATE.

Règle I3. Soit une entité E_j autorisée à gratifier à une entité E_m ($m \neq i$)

une SA de niveau L' avec paramètre-entité E_k. L'entité E_i propriétaire de E_k ne peut exécuter des actions $A_{ij/k}$ de type DELEGATE SA que si $L(SA) > L'$.

Règle I4. Une entité E_j qui s'est vue déléguée une SA de niveau L ne peut gratifier cette SA qu'aux entités E_m dont a_m^+ est de niveau $L' \geqslant L$, ce qui constitue une condition sur l'ensemble $\{ M \}$ de l'action DELEGATE/ABROGATE.

Les règles de cohérence mutuelle entre le SG et le DG.

Règle M1. Une entité E_i ne peut gratifier à une autre entité E_j le droit d'exécuter une SA de niveau L avec paramètre-entité E_k que si, dans le SG, E_i peut exécuter une SA de niveau $L' \geqslant L$ sur E_k.

Règle M2. Si, à l'initialisation dans le SG, une entité E_j peut exécuter une SA A_{jk} de niveau L, alors, dans le DG, une entité E_i ne peut gratifier à E_j le droit d'exécuter une action statique SA que si $L(SA) > L$.

Règle M3. A toute action $A_{ij/k}$ de type DELEGATE SA du DG doit correspondre, dans le SG, une action A_{ij} de type CREATE/DELETE. En fait, l'action DELEGATE SA avec paramètre-entité E_k ne peut être exécutée que par le propriétaire E_i de E_k et E_i doit donc être nécessairement autorisée à exécuter l'action CREATE/DELETE sur Ek.

Les règles de transformation du graphe dynamique.

Les règles de transformation du graphe dynamique ont pour but de montrer comment les droits d'exécuter des actions statiques et les droits de type GRANT/REVOKE s'écoulent dans un système d'autorisations, écoulements dus à l'exécution d'actions dynamiques.
La règle de transformation fournie ici est définie sur un groupe de trois

entités E_i, E_j et E_k et de deux actions dynamiques $A_{ij/z}$ et $A_{jk/z}$ adjacentes et définies sur la même entité-paramètre E_z et vise à contrôler le flux de droit sur l'entité E_z afin de surveiller l'état de protection de l'entité E_z.
Contrairement à ce qui se passe dans le graphe statique, l'action dynamique $A_{ik/z}$ peut ici ne pas exister. En effet, une action dynamique indirecte $A_{ik/z}$ n'existe que si la capacité d'exécuter $A_{jk/z}$ est subordonnée à l'exécution de l'action $A_{ij/z}$.

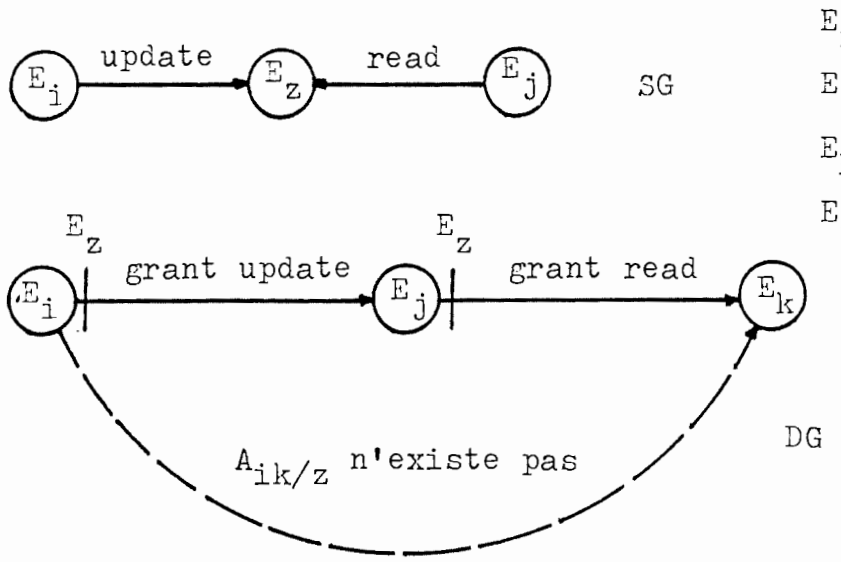
Définition.

On dit qu'une action $A_{jk/z}$ de type GRANT SA' est subordonnée à une action $A_{ij/z}$ si les deux conditions suivantes sont vérifiées :

1. Aucune action A_{jz} ne connecte directement E_j et E_z dans le graphe statique. ou
   si A_{jz} existe dans le graphe statique, $L(A_{jz}) < L(SA')$.

2. L'action statique SA" gratifiée ou déléguée par $A_{ij/z}$ est de niveau $L" > L(SA')$.

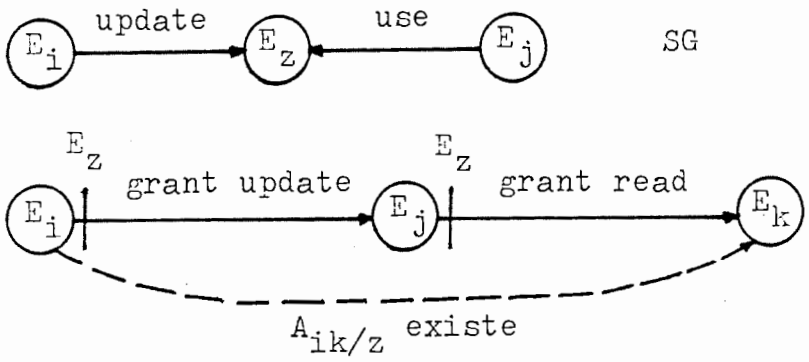
La figure 3.10. illustre le cas où une action indirecte n'existe pas. Cette figure montre que, dans le graphe statique, un programme d'application E_j peut lire le fichier de données E_z et, dans le graphe dynamique, gratifier cette action de lecture de E_z au programme d'application E_k; cette figure montre également que, dans le graphe statique, l'utilisateur E_i peut mettre à jour le fichier de données E_z et, dans le graphe dynamique, gratifier cette action de mise-à-jour de E_z à E_j.
L'action dynamique de gratification du droit de mettre à jour E_z entre E_i et E_j n'affecte pas la relation dynamique entre E_j et E_k car les deux actions de gratification $A_{ij/z}$ et $A_{jk/z}$ sont indépendantes l'une de l'autre et, dans ce cas l'action dynamique indirecte $A_{ij/z}$ n'existe pas.



E_i = utilisateur
E_j = programme d'application
E_k = programme d'application
E_z = fichier de données

Figure 3-10. Inexistence d'une action dynamique indirect.

La figure 3-11 montre un exemple d'action subordonnée où, dans le graphe statique, un utilisateur E_i a le droit de mettre à jour un fichier de données E_z et un programme d'application E_j a le droit d'utiliser le fichier de données E_z et où, dans le graphe dynamique, E_i a le droit de gratifier l'action de mise-à-jour de E_z à E_j et E_j a le droit de gratifier l'action de lecture de E_z au programme d'application E_k.
On peut vérifier aisément que ce schéma vérifie les conditions citées ci-dessus si bien que l'action $A_{jk/z}$ est subordonnée à l'action $A_{ij/z}$ et que, par conséquent, $A_{ik/z}$ existe.



E_i = utilisateur
E_j = programme d'application
E_k = programme d'application
E_z = fichier de données

Figure 3-11. Existence d'une action dynamique indirect.

Les actions subordonnées sont donc des actions GRANT particulières en ce sens qu'elles ne respectent pas la règle MI de mutuelle cohérence entre les graphes statique et dynamique.
Pour remédier à cette incohérence, on peut dire qu'une action subordonnée ne devient une action de gratification effective qu'après exécution de l'action

dynamique dont elle dépend.
Le concept d'action subordonnée introduit donc une notion de contrainte dépendant du temps.
La règle de transformation du graphe dynamique permet de déterminer l'action dynamique indirecte $A_{ij/z}$ de type GRANT SA à partir de l'action dynamique $A_{ij/z}$ de type DELEGATE SA" ou GRANT SA", de l'action dynamique $A_{jk/z}$ de type GRANT SA' subordonnée à $A_{ij/z}$, de l'entité E_i et de l'entité E_z :

$$SA = F(SA",SA',E_i,E_z).$$

Etant donnés la deuxième condition de la définition d'une action subordonnée et l'algorithme A, il est clair que, dans cet algorithme, on exécutera le point 1, si bien que SA = SA'. Ainsi, si nous revenons à la figure 3-11, il est clair que SA = USE si bien que $A_{ik/z}$ = GRANT USE.

Lorsque $A_{ij/z}$ et $A_{jk/z}$ sont conditionnées par des ensembles de prédicats $\{P_{ij/z}\}$ et $\{P_{jk/z}\}$, l'ensemble de prédicats $\{P_{ik/z}\}$ résultant est donné par $\{P_{ik/z}\} = \{P_{ij/z}\} \cup \{P_{jk/z}\}$.

Spécifications de la fonction de transformation du graphe dynamique.

Etant données la table de hiérarchisation des opérations, la table des classes de sécurité, deux actions adjascentes $A_{ij} \sim a_{ij}\ \{P_{ij}\}$ et $A_{jk} \sim a_{jk}\ \{P_{jk}\}$ s'exerçant entre trois entités E_i, E_j, E_k et portant sur la même entité-paramètre, soit E_p, la fonction de transformation du graphe dynamique détermine l'action indirecte résultante $A_{ik} \sim a_{ik}\ \{P_{ik}\}$ portant sur E_p, de la manière suivante :

- s'il existe dans le graphe statique une action A_{jp} de niveau hiérarchique supérieur ou égal à celui de a_{jk} ou si le niveau hiérarchique de a_{ij} est inférieur ou égal à celui de a_{jk} (négation des conditions d'action subordonnée), alors l'action résultante A_{ik} n'existe pas ($= \{\}$),

- sinon, l'action A_{ik} est composée de a_{jk} et de l'union des deux ensembles de prédicats $\{P_{ij}\}$ et $\{P_{jk}\}$.

Algorithme A' implémentant la fonction de transformation du graphe dynamique:

si ($\exists$ dans SG une action A_{jp} hiérarchiquement supérieure ou égale à a_{jk}) ou (a_{ij} est hiérarchiquement inférieur ou égal à a_{jk}).

alors $A_{ik} = \{\}$

sinon $A_{ik} = a_{jk} \sim \{P_{jk}\} \cup \{P_{ij}\}$

Cette règle de transformation ne peut être appliquée successivement à plus de trois entités et deux actions adjascentes que si ces entités et actions appartiennent à une même chaîne de GRANT (revoke).

Définition : Une chaîne de grant (revoke) sur une entité Ek est un chemin du graphe dynamique composé d'une action de type DELEGATE SA ou GRANT SA et d'actions subordonnées de type GRANT SA.
Une chaîne de grant (revoke) a donc les propriétés suivantes :

- Seule la première action de la chaîne peut être étiquetée par DELEGATE SA.

- Le niveau hiérarchique des SA déléguées ou gratifiées le long d'une telle chaîne ne peut augmenter (deuxième condition de la définition d'action subordonnée).

3.1.4. Utilisation du modèle.

Le modèle ACTEN permet d'une part d'analyser les états d'autorisation et de protection de chaque entité d'un système d'autorisations modélisé et d'autre part d'analyser la façon dont ces états peuvent évoluer.

3.1.4.1. Analyse des états d'autorisation et de protection.

Les états d'autorisation $Sa(E_i)$ et de protection $Sp(E_i)$ d'un système d'autorisations donné peuvent être déduits de l'analyse des graphes statique et dynamique ainsi que de la table des classes de sécurité. Pour déterminer $Sa(E_i)$, il faut tout d'abord examiner les graphes afin d'en retirer les actions directes de E_i (arcs partant du sommet E_i) et ensuite appliquer les règles de transformation afin d'en retirer les actions indirectes de E_i. Pour déterminer $Sp(E_i)$, il faut tout d'abord examiner les graphes afin d'en retirer les actions directes sur E_i (arcs arrivant au sommet E_i) et ensuite appliquer les règles de transformation afin d'en retirer les actions indirectes sur E_i.

Définition : Le sous-graphe direct d'analyse d'état d'autorisations (SGDA) d'une entité E_i est le sous-graphe statique ou dynamique composé du sommet E_i, des arcs partant de E_i et des sommets E_j où aboutissent ces arcs.

Définition : Le sous-graphe direct d'analyse d'état de protection (SGDP) d'une entité E_i est le sous-graphe statique ou dynamique composé du sommet E_i, des arcs aboutissant à E_i et des sommets E_j origines de ces arcs.

Définition : Une feuille d'un SGDA est un sommet duquel ne part aucun arc.

Définition : Une feuille d'un SGDP est un sommet auquel n'aboutit aucun arc.

Définition : On dit qu'une action $A \sim a \{P_A\}$ est hiérarchiquement supérieure à une action $A' \sim a' \{P_A'\}$ si ces deux actions sont toutes deux dynamiques ou toutes deux statiques, et

- soit l'opération a ou l'action a (cas d'une action A dynamique) est de niveau hiérarchique supérieur à celui de l'opération a' ou de l'action a' (cas d'une action A'

dynamique) et l'ensemble des prédiats $\{P_A\}$ est inclus ou égal à l'ensemble des prédicats $\{P_A'\}$.

- soit l'opération a ou l'action a (cas d'une action A dynamique) est de niveau hiérarchique égal à celui de l'opération a' ou de l'action a' (cas d'une action A' dynamique) et l'ensemble des prédicats $\{P_A\}$ est strictement inclus dans l'ensemble des prédicats $\{P_A'\}$.

Définition : On dit qu'une action $A \sim a \{P_A\}$ est hiérarchiquement inférieure à une action $A' \sim a' \{P_A'\}$ si ces deux actions sont toutes deux dynamiques ou toutes deux statiques, et

- soit l'opération a ou l'action a (cas d'une action A dynamique) est de niveau hiérarchique inférieur à celui de l'opération a' ou de l'action a' (cas d'une action A' dynamique) et l'ensemble des prédicats $\{P_A\}$ comprend ou est égal à l'ensemble des prédicats $\{P_A'\}$.

- soit l'opération a ou l'action a (cas d'une action A dynamique) est de niveau hiérarchique égal à celui de l'opération a' ou de l'action a' (cas d'une action A' dynamique) et l'ensemble des prédicats $\{P_A\}$ comprend sans être égal à l'ensemble des prédicats $\{P_A'\}$.

Définition : On dit qu'une action $A \sim a \{P_A\}$ est hiérarchiquement égale à une action $A' \sim a' \{P_A'\}$ si ces deux actions sont toutes deux dynamiques ou toutes deux statiques, et si l'opération a ou l'action a (cas d'une action A dynamique) est de niveau hiérarchique égal à celui de l'opération a' ou de l'action a' (cas d'une action A' dynamique) et l'ensemble des prédicats $\{P_A\}$ est égal à l'ensemble des prédicats $\{P_A'\}$.

Définition : On dit qu'une action $A \sim a \{P_A\}$ est hiérarchiquement incomparable à une action $A' \sim a' \{P_A'\}$ si A n'est ni hiérarchiquement supérieur, ni hiérarchiquement inférieure, ni hiérarchiquement égale à A'.

Définition : On appelle liste implicite d'actions exécutables sur (ou par) une entité E_i une liste d'actions sur (ou par) E_i telle qu'aucune action n'y est hiérarchiquement supérieure ou égale à aucune autre de la liste, telle que toute action directe ou indirecte contenue dans les graphes du système est présente directement ou indirectement (via une action hiérarchiquement supérieure) dans cette liste et telle que toute action de cette liste est directement ou indirectement (action indirecte) présente dans l'un des graphes du système.

Définition : On appelle liste implicite partielle d'actions exécutables sur (ou par) une entité E_i une liste d'actions sur (ou par) E_i telle qu'aucune action n'y est hiérarchiquement supérieure ou égale à aucune autre de la liste et telle que toute action de cette liste est directement ou indirectement (action indirecte) présente dans l'un des graphes du système.

Spécifications de la fonction de détermination d'un état d'autorisation.

Etant donnés les graphes statique et dynamique, la table de hiérarchisation des opérations, la table de classes de sécurité, la fonction de transformation du graphe statique implémentée par l'algorithme A, la fonction de transformation du graphe dynamique implémentée par l'algorithme A', et les définitions exposées ci-dessus, la fonction de détermination de l'état d'autorisation d'une entité E_i donnée a pour objectif de créer la liste implicite d'actions exécutables par E_i.

Avant de passer à l'algorithme B implémentant cette fonction, il est à noter que cet algorithme fait appel à un autre algorithme B1 construit de manière récursive et résolvant un sous-problème auxiliaire répondant aux spécifications suivantes : étant donnés les graphes statique et dynamique, la table de hiérarchisation des opérations, la table de classes de sécurité, la fonction de transformation du graphe statique implémentée par l'algorithme A, la fonction de transformation du graphe dynamique implémentée par l'algorithme A' et les définitions exposées ci-dessus, le sous-problème auxiliaire a pour objectif d'ajouter à la liste implicite partielle courante d'actions exécutables par une entité E_i toute action indirecte ou directe obtenue en imposant la première action (qui peut déjà

être une action indirecte) des chaînes de détermination des actions indirectes, tout en maintenant la définition de liste implicite partielle d'actions exécutables par E_i.

Algorithme B (E_i)

début
pour toute action A_{ij} du SGDA statique de E_i faire
début
A = A_{ij}
Bl (A, A_{ij}, SG)
fin

pour tout action A_{ij} du SGDA dynamique de E_i faire
début
A = A_{ij}
Bl (A, A_{ij}, DG)
fin

fin

n.b. : A = première action (dircte ou indirecte) couramment imposée.

Algorithme Bl (A, A_{xy}, graphe)

début
si E_{ij} n'est pas une feuille du SGDA de E_x dans graphe
alors pour toute action A_{yk} du SGDA de E_y dans graphe faire
si graphe = SG
alors Bl (résultat algo. A (A, A_{yk}), A_{yk}, graphe)
sinon Bl (résultat algo A' (A, A_{yk}), A_{yk}, graphe)

si il n'existe dans autorisations (E_i) aucune action hiérarchiquement supérieure ou égale à A

alors début
  pour toute action A_{im} de autorisations (E_i) faire
    si A_{im} est de niveau hiérarchique inférieur à A
    alors effacer A_{im} de autorisations (E_i)
  ajouter A dans autorisations (E_i)
  fin
fin

n.b. autorisations (E_i) = liste implicite d'actions de E_i.

Spécifications de la fonction de détermination d'un état de protection.

Etant donnés les graphes statique et dynamique, la table de hiérarchisation des opérations, la table de classes de sécurité la fonction de transformation du graphe statique implémentée par l'algorithme A, la fonction de transformation du graphe dynamique implémentée par l'algorithme A, la fonction de transformation du graphe dynamique implémentée par l'algorithme A', et les définitions exposées ci-dessus, la fonction de détermination de l'état de protection d'une entité E_i donnée a pour objectif de créer la liste implicite d'actions exécutables sur E_i.

Avant de passer à l'algorithme B' implémentant cette fonction. il est à noter que cet algorithme fait appel à un autre algorithme B2 construit de manière récursive et résolvant un sous-problème auxiliaire répondant aux spécifications suivantes : étant donnés les graphes statique et dynamique, la table de hiérarchisation des opérations. la table de classes de sécurité, la fonction de transformation du graphe statique implémentée par l'algorithme A, la fonction de transformation du graphe dynamique implémentée par l'algorithme A', et les définitions exposées ci-dessus, le sous-problème auxiliaire a pour objectif d'ajouter à la liste implicite partielle courante d'actions exécutables sur une entité E_i toue action indirecte ou directe obtene en imposant la dernière action (qui peut déjà être une action indirecte) des chaînes de détermination des actions indirectes, tout en maintenant la définition de liste implicite partielle d'actions exécutables sur E_i

Algorithme B' (E_i)

début
pour toute action Aji du SGDP statique de E_i faire
début
A = A_{ji}
B2 (A, A_{ji}, SG)
fin

pour toute action A_{ji} du SGDP dynamique de E_i faire

début
A = Aji
B2 (A, Aji, DG)
fin

fin

n.b.: A = dernière action (directe ou indirecte) couramment imposée.

Algorithme B2 (A, A_{yx}, graphe)

début
si E_y n'est pas une feuille du SGDP de E_x dans graphe alors pour toute action A_{ky} du SGDP de E_y dans graphe faire
si graphe = SG
alors B2 (résultat algo. A (A_{ky}, A), A_{ky}, graphe)
sinon B2 (résultat algo A' (A_{ky}, A), A_{ky}, graphe)

si il n'existe dans protections (E_i) aucune action hiérarchiquement supérieure ou égale à A
alors début
pour toute action A_{mi} de protections (Ei) faire
si A_{mi} est de niveau hiérarchique inférieur à A
alors effacer A_{mi} de protections (E_i)
ajouter A dans protections (E_i)
fin

fin

n.b. protections (E_i) = liste implicite d'actions sur E_i.

3.1.4.2. Analyse de l'évolution des états d'autorisation et de protection.

La phase d'analyse de l'évolution des états d'autorisation et de protection d'une entité E_i a pour but de déterminer les intervalles de variation de $Sa(E_i)$ et $Sp(E_i)$.
Il s'agit donc d'une part, de déterminer les états d'autorisation minimum et maximum $Sa(E_i)$, et d'autre part, de déterminer les états de protection minimum et maximum $Sp(E_i)$.

N.B. Que l'état de protection d'une entité E_i est minimum ne signifie pas qu'un maximum d'entités ont un maximum de droits d'actions sur E_i mais bien qu'un minimum d'entités ont un minimum de droits d'actions sur E_i, conformément à ce qui a été dit au point 3.1.1.4.
Que l'état de protection d'une entité E_i est maximum ne signifie pas qu'un minimum d'entités ont un minimum de droits d'actions sur E_i mais bien qu'un maximum d'entités ont un maximum de droits d'actions sur E_i, conformément à ce qui a été dit au point 3.1.1.4.

Afin de déterminer $Sa(E_i)$ minimum et $Sp(E_i)$ minimum, considérons que toutes les actions dynamiques de types ABROGATE et REVOKE sont exécutées et ce, quels que soient leurs prédicats associés. Effectuons ensuite l'analyse des

états d'autorisation et de protection de E_i telle que décrite au point 3.1.4.1.
Afin de déterminer $Sa(E_i)$ maximum et $Sp(E_i)$ maximum, considérons que toutes les actions dynamiques de types DELEGATE et GRANT sont exécutées et ce, quels que soient leurs prédicats associés.
Effectuons ensuite l'analyse des états d'autorisation et de protection de E_i telle que décrite au point 3.1.4.1.

Un exemple de modélisation d'un système d'autorisations effectuée à l'aide du modèle ACTEN sera discuté ultérieurement dans l'étude d'applicabilité du modèle (point 3.4.).
Le problème de sécurité du modèle ACTEN sera discuté au point 3.5. du présent travail.

3.2. DISCUSSION DU MODELE

Le présent paragraphe vise à apporter certaines précisions à la présentation du modèle ACTEN ci-dessus. Ces précisions concernent :

- la règle de transformation du graphe statique,
- l'application de la règle de transformation du graphe statique
- l'algorithme d'analyse des états d'autorisation et de protection.

3.2.1. La règle de transformation du graphe statique (algorithme A)

L'algorithme de détermination de l'action statique indirecte donné au point 3.1.3.1. peut être caractérisé de redondant.
En effet, cet algorithme ne tient pas compte de toute la puissance des règles de cohérence interne du graphe statique : de telles règles assurent qu'à tout moment, pour toutes entités E_i et E_j, on a $L(a_i^+) \geqslant L(a_{ij})$, si bien que certains tests effectués sont inutiles. Il s'ensuit que l'on peut directement comparer $L(a_{jk})$ et $L(a_i+)$.

On obtient ainsi l'algorithme suivant :

début
si L(ajk) > L(ai+)
alors Aik = $a_i^+ \sim (\{p_{ij}\} \wedge \{p_{jk}\})$
sinon Aik = $a_{jk} \sim (\{p_{ij}\} \wedge \{p_{jk}\})$
fin

3.2.2. L'application répétée de la règle de transformation du graphe statique.

Nous avons vu au point 3.1.3.1. que lorsque deux entités s'interposent entre E_i et E_k, le droit indirect A_{ik} est déterminé en appliquant deux fois la règle de transformation: la figure 3-12 illustre une telle disposition où E_i est reliée à E_j par une action A_{ij}, E_j est reliée à E_l par une action A_{jl} et E_l est reliée à E_k par une action A_{lk}.

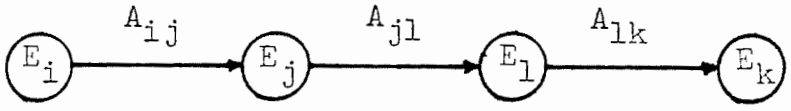


Figure 3-12. Disposition à deux entités interposées

La détermination du droit indirect A_{ik} telle qu'indiquée au point 3.1.3.1. se fait en deux étapes :

I- considérer E_i, E_j, E_l, A_{ij}, A_{jl} et appliquer la règle de transformation pour déterminer A_{il} :

2- considérer E_i, E_l, E_k, A_{il}, A_{ik} et appliquer la règle de transformation pour déterminer A_{ik}.

Cette façon de faire est illustrée par la figure 3-13.

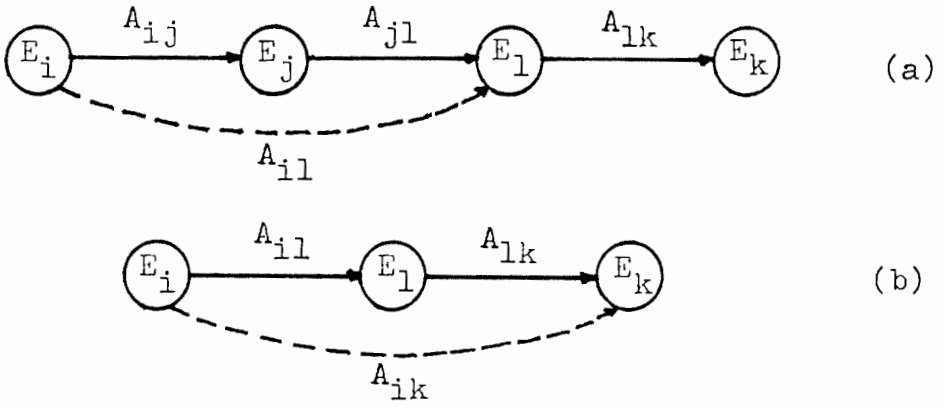


Figure 3-13. Première façon de déterminer A_{ik}

Une autre façon de faire est la suivante :

1- considérer E_j, E_l, E_k, A_{jl}, A_{lk} et appliquer la règle de transformation pour déterminer A_{jk}.

2- considérer E_i, E_j, E_k, A_{ij}, A_{jk}, et appliquer la règle de transformation pour déterminer A_{ik}.

Cette deuxième façon de faire est illustrée par la figure 3-14.

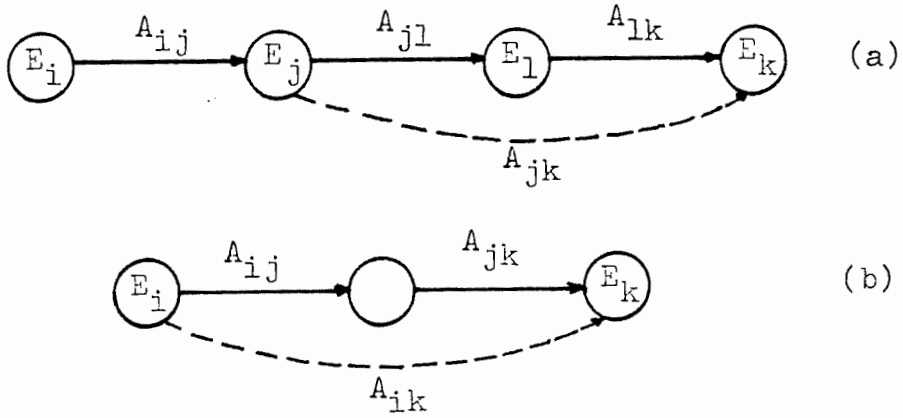


Figure 3-14. Deuxième façon de déterminer A_{ik}

Le prédicat résultant est dans les deux cas $\{p_{ik}\} = \{p_{ij}\} \wedge \{p_{jk}\}$.

A priori, rien ne permet d'affirmer que ces deux façons de faire aboutissent au même résultat; en fait, on peut même trouver aisément des exemples où les deux façons de faire donnent deux résultats différents. La figure ~~3-15~~ 3-15 est un exemple de détermination où la première façon de déterminer résulte en une action dont l'opération est write et la deuxième façon en une action dont l'opération est update.
On peut démontrer que la première méthode aboutit toujours à une action $A_{ik}^{(I)}$ de niveau hiérarchique supérieur ou égal à $A_{ik}^{(2)}$ trouvé par la seconde méthode. (cfr démonstration en annexe A).

Si l'on se place du côté possesseur d'une entité, il semble que la deuxième façon de faire est la plus indiquée car limitant beaucoup plus les droits d'accès indirects sur l'entité possédée.

Cependant, la première façon de faire garantit une meilleure fluidité du système vis-à-vis de la propagation de droits.

Le choix d'une de ces deux méthodes traduit donc la philosophie que l'on veut inculquer au système.

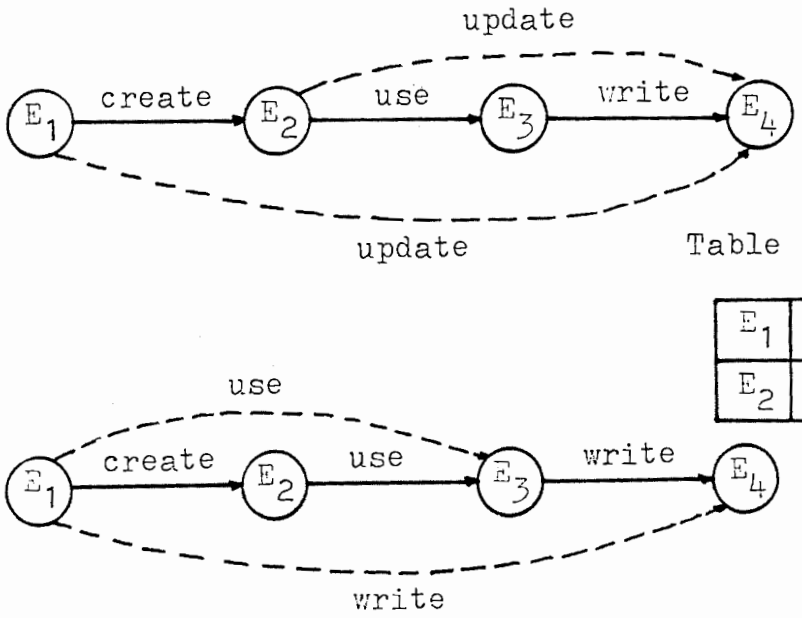


Table de classes de sécurité

E_1	+ create/delete
E_2	+ update

FIGURE 3-15. Exemple où les deux méthodes donnent des résultats différer

3.2.3. L'algorithme d'analyse des états d'autorisation et de protection

Si l'on se réfère au point 3.2.2., on peut dire que l'algorithme d'analyse des états d'autorisation et de protection fait apparaître la première façon de déterminer l'action indirecte dans son analyse de l'état d'autorisation d'une entité et fait apparaître la seconde façon dans son analyse de l'état de protection d'une entité.

Une telle incohérence ne peut subsister et un choix de méthode doit être fait.

Choix d'une méthode de détermination des actions indirectes.

La détermination des états d'autorisation de toutes les entités d'un système d'autorisations permet de déduire immédiatement les états de protection de toutes ces entités, et vice versa.

Il s'ensuit que l'on peut oublier une des deux analyses faites par l'algorithme. Convenons de retenir l'analyse qui implémente la méthode de détermination des actions indirectes que nous allons choisir.

Ce choix de méthode penche ici vers la deuxième méthode car elle protège davantage le possesseur d'entité; on peut répondre à l'argument de meilleure fluidité du système favorisant la première méthode que tout droit d'action indirect peut toujours être remplacé par un droit d'action direct que l'on pourra demander au possesseur, si bien que cet argument paraît moindre que celui avancé en faveur de la deuxième méthode.

Il découle de ce choix que l'analyse retenue est celle des états de protection des entités.

3.3. EN QUOI LE MODELE A C T E N PEUT-IL ETRE INCLUS DANS LE MODELE GENERAL PRESENTE PAR E. L. LEISS ?

Nous avons dit au point 2.1. que le modèle de système d'autorisations présenté par E. L. Leiss était un modèle général au sens où il est capable de décrire la plupart des systèmes d'autorisations couramment utilisés dans les systèmes informatiques actuels.

Le modèle général de systèmes d'autorisations est-il capable de décrire les systèmes d'autorisations issus du modèle conceptuel ACTEN ?
Afin de répondre à cette question, nous allons passer en revue chaque concept du modèle ACTEN en nous demandant si ce concept est représentable dans le modèle général sans y introduire d'outils supplémentaires.

Le concept d'entité du modèle ACTEN peut être représenté par les objets et sujets du modèle général; le concept d'opération de base par un droit de base du modèle général: le concept de direction d'une action est représenté implicitement dans la matrice d'accès.

Le type de prédicat portant sur l'existence d'une action ne peut en toute généralité pas être inclus dans le modèle général car celui-ci ne fait pas état de possibilité d'introduire des conditions préalables à l'exécution d'un droit. Une solution factice pour insérer cette notion dans le modèle général serait :

1- introduire ce type de prédicat en tant qu'élément de test dans une commande insérant un droit x correspondant à l'action sur laquelle porte le prédicat
2- mémoriser que ce droit x est dépendant du droit y sur lequel porte le test de la commande
3- ajouter dans toute commande comportant une primitive de suppression du droit y une primitive de suppression du droit x.

Cette solution est en effet factice d'un point de vue pratique, car le prédicat porte sur une action qui peut comporter un prédicat portant sur l'existence de la première action si bien qu'aucune commande ne pourrait

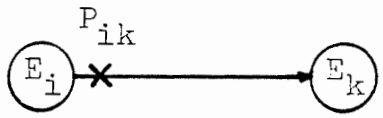
jamais insérer un droit correspondant à l'une de ces actions si aucun de ces droits ne se trouve initialement dans la matrice d'accès.

Insertion de d_{ij} P_{ij}	Insertion de d'_{kl} P_{kl}
<u>if</u> d' in (x_k,x_l)	<u>if</u> d in (x_i,x_j)
<u>then</u> enter d into (x_i,x_j)	<u>then</u> enter d' into (x_k,x_l)

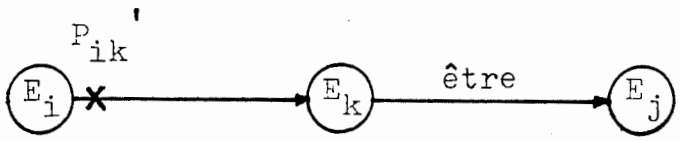
Le problème serait praticable si un prédicat ne pouvait porter que sur l'existence d'une opération de base et non plus sur l'existence d'une action.

Le type de prédicat portant sur l'inclusion de la valeur d'un attribut d'une entité E_i dans un ensemble de valeurs peut être modélisé dans le modèle général en transformant cet attribut en une entité E_j et en représentant l'existence de cette inclusion par un droit "être" de E_i vers E_j.

<u>Exemple</u> : P_{ik} = "si le périphérique E_k est de type x"



devient



où P'_{ik} = "si le périphérique E_k a le droit "être" sur E_j".

Le type de prédicat portant sur l'exécution effective d'une action ainsi que le type de prédicat comportant une condition temporelle peuvent subir un traitement identique (transformation d'un attribut en entité) pour être modélisés dans le modèle général de E.Leiss.

La table de classes de sécurité peut être représentée dans le modèle

général en introduisant un objet fictif servant pour tout sujet à représenter dans la matrice d'accès sa potentialité active et un sujet fictif servant pour tout sujet et objet à représenter sa potentialité passive.

La table de hiérarchisation peut être définie implicitement dans l'ensemble des commandes du modèle général: l'insertion d'un droit en une case de la matrice d'accès ne se fera que si cette case ne contient pas déjà un droit de niveau plus élevé non conditionné par un ensemble de prédicats incluant l'ensemble des prédicats conditionnant le droit que l'on veut insérer: une commande testant l'existence d'un droit d avant insertion d'un nouveau droit se traduira par plusieurs commandes testant chacune l'existence d'un droit de niveau plus élevé ou égal au niveau du droit d.
Il est donc théoriquement possible de représenter la notion de hiérarchisation dans le modèle général.

La notion de possesseur d'entité pourrait être représentée par un droit "détenir" auquel le niveau hiérarchique le plus élevé serait affecté.
La table d'états d'entités pourrait ainsi être représentée dans la matrice d'accès elle-même.

Les règles de cohérence et de transformation pourraient être exprimées à l'aide de commandes particulières du modèle général, commandes particulières en ce sens que toute terminaison d'exécution d'une commande non particulière déclencherait l'exécution de ces commandes particulières.

On peut donc dire que le modèle général peut représenter les systèmes d'autorisations issus du modèle ACTEN et semble répondre à son qualificatif de modèle général.

3.4. ETUDE DE L'APPLICABILITE DU MODELE A C T E N.

Il est clair que le modèle est inutile pour modéliser des structures dont les éléments ne sont pas des ressources à protéger ni des ressources dont on veut limiter l'action car l'on n'envisagera pas de système de protection pour ces structures "anarchiques" où tous les éléments agissent à leur guise.

Est-ce que le modèle ACTEN permet de modéliser n'importe quelle structure dont les éléments sont des ressources que l'on désire protéger ou des ressources dont on veut limiter l'action ?
Peut-on modéliser les besoins de sécurité d'une structure hospitalière, d'une entreprise, d'une bibliothèque, ...?

Nous allons ci-dessous modéliser les besoins de sécurité d'une structure hospitalière.

3.4.1. Quelles sont les ressources impliquées dans les besoins de sécurité ?

Les ressources impliquées sont : un chef d'hôpital. un médecin pédiatre, un médecin chirurgien, un médecin consultant, quatre infirmières, quatre lits,une femme d'ouvrage, trois patients, deux locaux, deux chambres, deux armoires, deux stocks de médicaments et deux visiteurs.

- chef-hôpital
- médecin-pédiatre
- médecin-chirurgien
- médecin-consultant
- infirmière-1
- infirmière-2
- infirmière-3
- infirmière-4
- lit-1
- lit-2
- lit-3
- lit-4
- femme-ouvrage
- patient-1
- patient-2
- patient-3
- local-1
- local-2
- chambre-1
- chambre-2
- armoire-1
- armoire-2
- médicament-1
- médicament-2
- visiteur-1
- visiteur-2

3.4.2. Quels sont les besoins de sécurité à modéliser ?

a. Fixons l'image de la structure à un moment donné.

Le médecin-pédiatre peut soigner un patient à condition que ce patient soit un enfant.

Le médecin-pédiatre peut opérer un patient à condition que ce patient soit un enfant et qu'il soit assisté du médecin-chirurgien.

Le médecin-pédiatre peut accéder à tout local.

Le médecin-pédiatre peut accéder à toute chambre contenant le lit d'un patient qu'il soigne.

Le médecin-chirurgien peut opérer un patient.

Le médecin-chirurgien peut accéder à tout local.

Le médecin-chirurgien peut accéder à toute chambre contenant le lit d'un patient opéré le jour même.

Le médecin-consultant peut consulter un patient.

Le médecin-consultant peut accéder à tout local ne contenant pas d'armoire à médicament-2.

Le médecin-consultant peut accéder à toute chambre.

L'infirmière-1 peut soigner un patient à condition que ce patient soit un enfant.

La femme d'ouvrage peut nettoyer tout local ne contenant pas d'armoire à médicament-2.

La femme d'ouvrage peut nettoyer toute chambre ne contenant pas le lit d'un patient opéré le jour même.

Un patient ne peut accéder à une chambre que si elle comprend son propre lit.

b. Evolution possible de la structure.

Le chef-hôpital peut gratifier/reprendre au médecin-consultant le droit d'accès à tout local.

Le chef-hôpital peut gratifier/reprendre au médecin-chirurgien le droit d'opérer.

Le chef-hôpital peut gratifier/reprendre au médecin-consultant le droit de consulter.

Le chef-hôpital peut gratifier/reprendre au médecin-pédiatre le droit d'opérer un patient enfant avec l'aide du médecin-chirurgien.

Le chef-hôpital peut gratifier/reprendre à un local le droit de contenir une armoire.

Le chef-hôpital peut gratifier/reprendre à une armoire le droit de contenir un type de médicament.

Le chef-hôpital peut gratifier/reprendre à une chambre le droit de contenir un lit.

Le chef-hôpital peut déléguer/abroger à une médecin-pédiatre le droit de soigner le patient-1 avec option de gratification pour l'infirmière-1.

Le chef-hôpital peut déléguer/abroger au médecin-chirurgien le droit de soigner le patient-2 avec option de gratification pour l'infirmière-2.

Le chef-hôpital peut déléguer/abroger au médecin-chirurgien le droit de soigner le patient-3 avec option de gratification pour l'infirmière-2 et pour l'infirmière-3.

Un patient peut gratifier/reprendre à un visiteur le droit d'accéder à la chambre contenant son propre lit.

c. Hiérarchisation des opérations.

Dans les opérations statiques, on distingue la hiérarchisation suivante :

L(opérer) > L(soigner) > L(consulter), L(nettoyer) > L(accéder).

Dans les opérations dynamiques. on distingue la hiérarchisation suivante

L(déléguer/abroger) > L(gratifier/reprendre).

d. Contrainte sur les entités (table de classes de sécurité).

Le "Chef-hôpital" peut au maximum "accéder" et ne peut rien subir.
Le "Médecin-pédiatre" peut au maximum "opérer" et ne peut rien subir.
Le "Médecin-chirurgien" peut au maximum "opérer" et ne peut rien subir.
Le "Médecin-consultant" peut au maximum "soigner" et ne peut rien subir.
L' "Infirmière-1" peut au maximum "soigner" et ne peut rien subir.
L' "Infirmière-2" peut au maximum "soigner"et ne peut rien subir.
L' "Infirmière-3" peut au maximum "soigner" et ne peut rien subir.
L' "Infirmière-4" peut au maximum "soigner" et ne peut rien subir.
La "Femme-ouvrage" peut au maximum "nettoyer" et ne peut rien subir.
Le "Patient-1" peut au maximum "accéder" et peut être "opéré" au maximum.
Le "Patient-2" peut au maximum "accéder" et peut être "opéré" au maximum.
Le "Patient-3" peut au maximum "accéder" et peut être opéré" au maximum.

Le "Local-1" peut au maximum "contenir" et peut être "nettoyé" au maximum.
Le "Local-2" peut au maximum "contenir" et peut être "nettoyé" au maximum.

L' "Armoire-1" peut au maximum "contenir" et peut être "contenue" au maximum.
L' "Armoire-2" peut au maximum "contenir" et peut être "contenue" au maximum.
Le "Médicament-1" ne peut rien faire et peut être "contenu"
Le "Médicament-2" ne peut rien faire et peut être "contenu" au maximum.
Le "Visiteur-1" peut au maximum "accéder" et ne peut rien subir.
Le "Visiteur-2" peut au maximum "accéder" et ne peut rien subir.
La "Chambre-1" peut au maximum "contenir" et peut être "nettoyée" au maximum.
La "Chambre-2" peut au maximum "contenir" et peut être "nettoyée" au maximum.
Le "Lit-1" ne peut rien faire et peut être "contenu" au maximum.
Le "Lit-2" ne peut rien faire et peut être "contenu" au maximum.
Le "Lit-3" ne peut rien faire et peut être "contenu" au maximum.
Le "Lit-4" ne peut rien faire et peut être "contenu" au maximum.

e. Possession d'entités (table d'états d'entités).

L'entité "Chef-hôpital" possède les entités "patient-1", "patient-2", "patient-3".
L'entité "Chef-hôpital" possède les entités "chambre-1", "chambre-2", "local-1".
L'entité "Chef-hôpital" possède les entités "local-2", "armoire-1", "armoire-2".
L'entité "Chef-hôpital" possède les entités "lit-1", "lit-2", "lit-3", "lit-4".

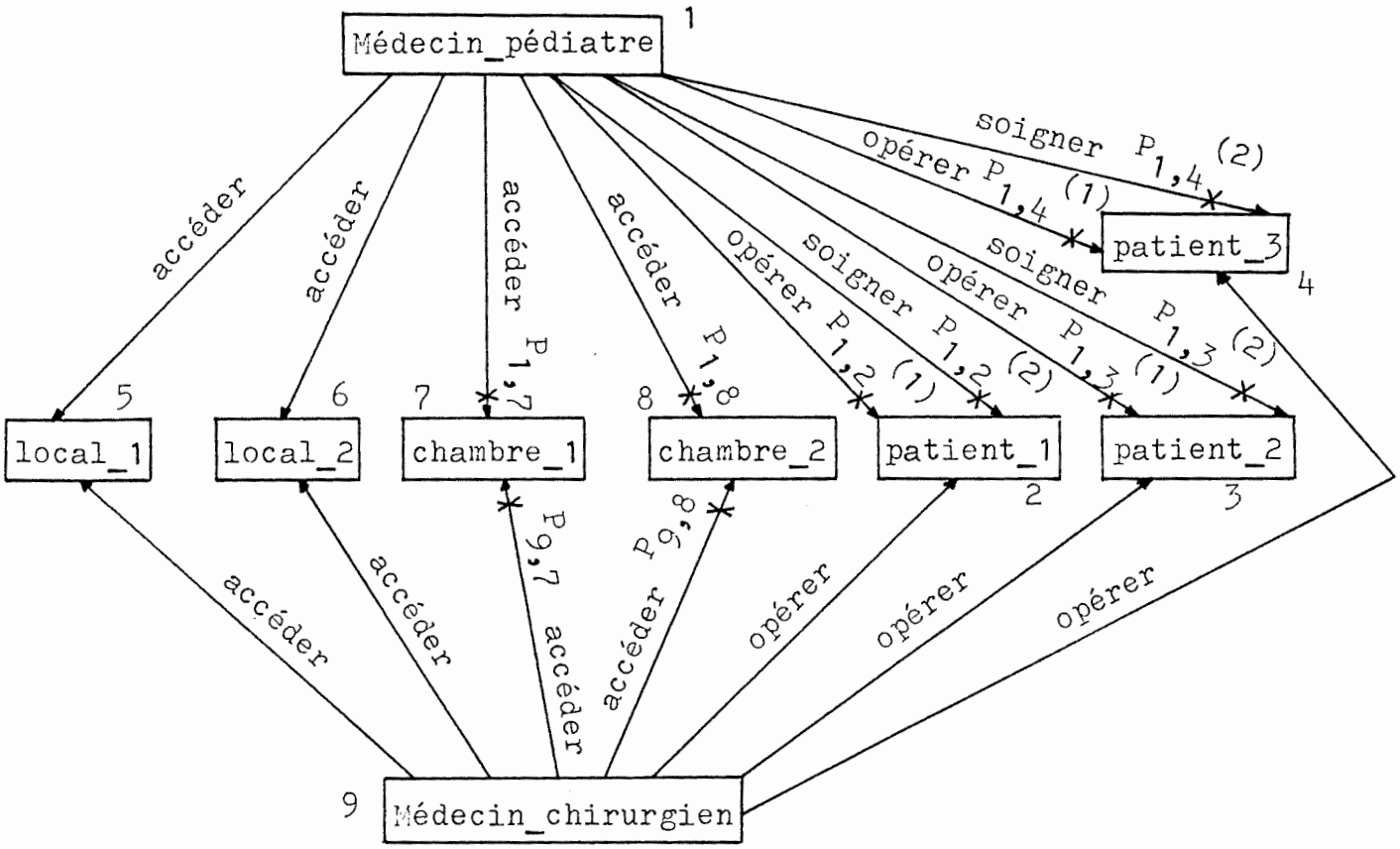
3.4.3. Modélisation.

Dans un premier temps, nous allons traduire l'image fixe de la structure dans le graphe statique, l'évolution possible de la structure dans le graphe dynamique, la hiérarchisation des opérations dans la table de hiérarchisation, les contraintes sur les entités dans la table de classes de sécurité, les possessions d'entités dans la table d'états d'entités.

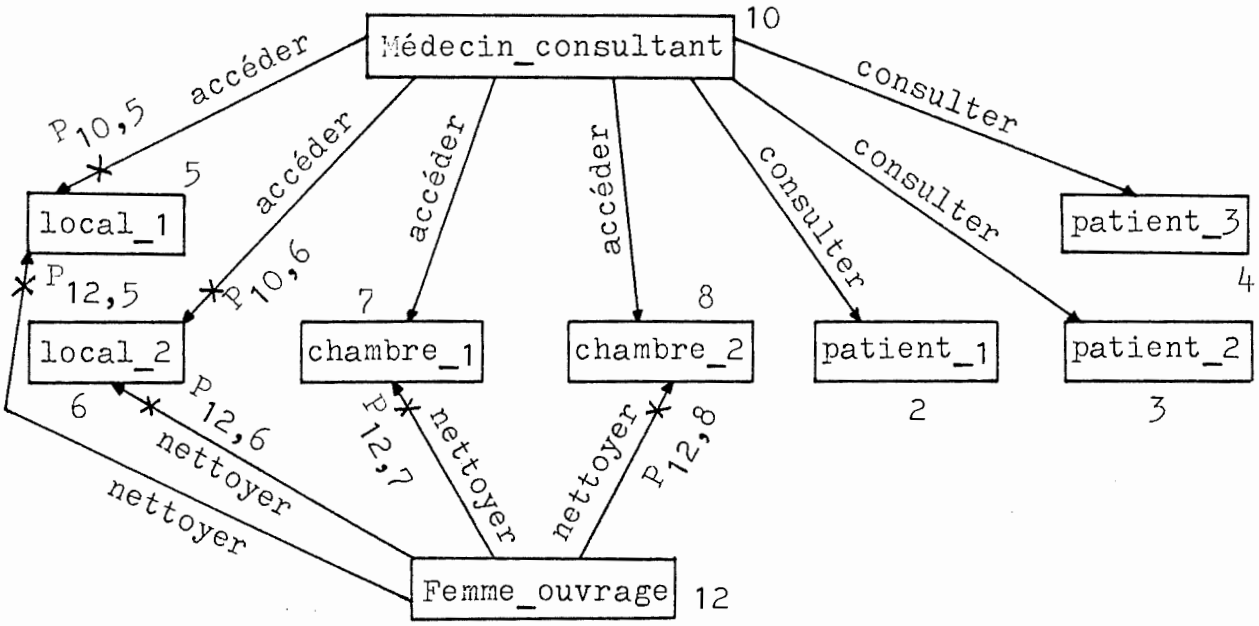
Dans un second temps, nous vérifierons si la modélisation ainsi obtenue vérifie les règles de conformité du modèle et si besoin nous adapterons cette modélisation afin de respecter ces règles de conformité.

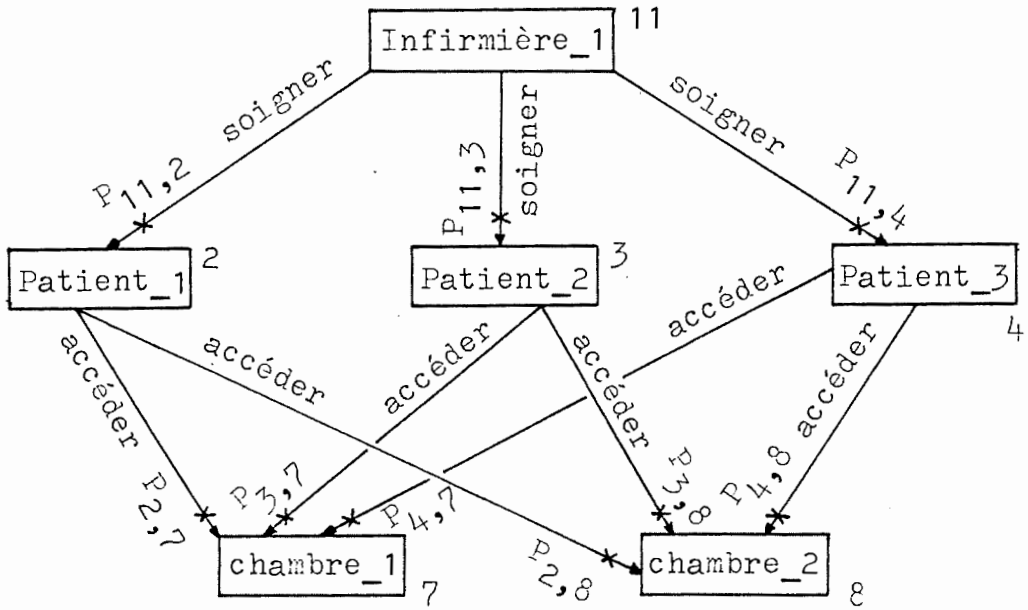
L'expression des graphes a été décomposée en plusieurs parties pour des raisons de lisibilité et les redondances apparaissant dans ces trois graphes ne sont dues qu'à celà.

a. Expression du graphe statique SG.



N.B. Les numéros apparaissant aux côtés des entités ne sont qu'une numérotation identificatrice de ces entités.


Médecin_consultant
10
accéder
consulter
$P_{10,5}$
$P_{10,6}$
$P_{12,5}$
$P_{12,6}$
$P_{12,7}$
$P_{12,8}$
nettoyer
local_1
5
local_2
6
chambre_1
7
chambre_2
8
patient_1
2
patient_2
3
patient_3
4
Femme_ouvrage
12


Infirmière_1
11
soigner
$P_{11,2}$
$P_{11,3}$
$P_{11,4}$
Patient_1
2
Patient_2
3
Patient_3
4
accéder
$P_{2,7}$
$P_{3,7}$
$P_{4,7}$
$P_{2,8}$
$P_{3,8}$
$P_{4,8}$
chambre_1
7
chambre_2
8

Expression des contraintes.

$\{P^{(1)}_{1,2}\}$ = si patient-1 est un enfant et
si médecin-chirugien use effectivement de son droit d'opérer patient-1

$\{P^{(2)}_{1,2}\}$ = si patient-1 est un enfant.

$\{P^{(1)}_{1,3}\}$ = si patient-2 est un enfant et
si médecin-chirurgien use effectivement de son droit d'opérer patient-2.

$\{P^{(2)}_{1,3}\}$ = si patient-2 est un enfant.

$\{P^{(1)}_{1,4}\}$ = si patient-3 est un enfant et
si médecin-chirurgien use effectivement de son droit d'opérer patient-3.

$\{P^{(2)}_{1,4}\}$ = si patient-3 est un enfant.

$\{P_{1,7}\}$ = si $\exists$ x,y : médecin-pédiatre peut soigner x et chambre-1 peut contenir y et y est le lit de x.

$\{P_{1,8}\}$ = si $\exists$ x,y : médecin-pédiatre peut soigner x et chambre-2 peut contenir y et y est le lit de x.

$\{P_{2,7}\}$ = si $\exists$ x : chambre-1 peut contenir x et x est le lit de patient-1.

$\{P_{2,8}\}$ = si $\exists$ x : chambre-2 peut contenir x et x est le lit de patient-1.

$\{P_{3,7}\}$ = si $\exists$ x : chambre-1 peut contenir x et x est le lit de patient-2.

$\{P_{3,8}\}$ = si $\exists$ x : chambre-2 peut contenir x et x est le lit de patient-2.

$\{P_{4,7}\}$ = si $\exists$ x : chambre-1 peut contenir x et x est le lit de patient-3.

$\{P_{4,8}\}$ = si $\exists$ x : chambre-2 peut contenir x et x est le lit de patient-3.

$\{P_{9,7}\}$ = si $\exists$ x,y : chambre-1 peut contenir x et x est le lit de y et y a été opéré le jour même.

$\{P_{9,8}\}$ = si $\exists$ x,y : chambre-2 peut contenir x et x est le lit de y et y a été opéré le jour même.

$\{P_{10,5}\}$ = si $\nexists$ x : local-1 peut contenir x et x peut contenir médicament-2.

$\{P_{10,6}\}$ = si $\nexists$ x : local-2 peut contenir x et x peut contenir médicament-2.

$\{P_{11,2}\}$ = si patient-1 est un enfant.

$\{P_{11,3}\}$ = si patient-2 est un enfant.

$\{P_{11,4}\}$ = si patient-3 est un enfant.

$\{P_{12,5}\}$ = si $\nexists$ x : local-1 peut contenir x et x peut contenir médicament-2.

$\{P_{12,6}\}$ = si $\nexists$ x : local-2 peut contenir x et x peut contenir médicament-2.

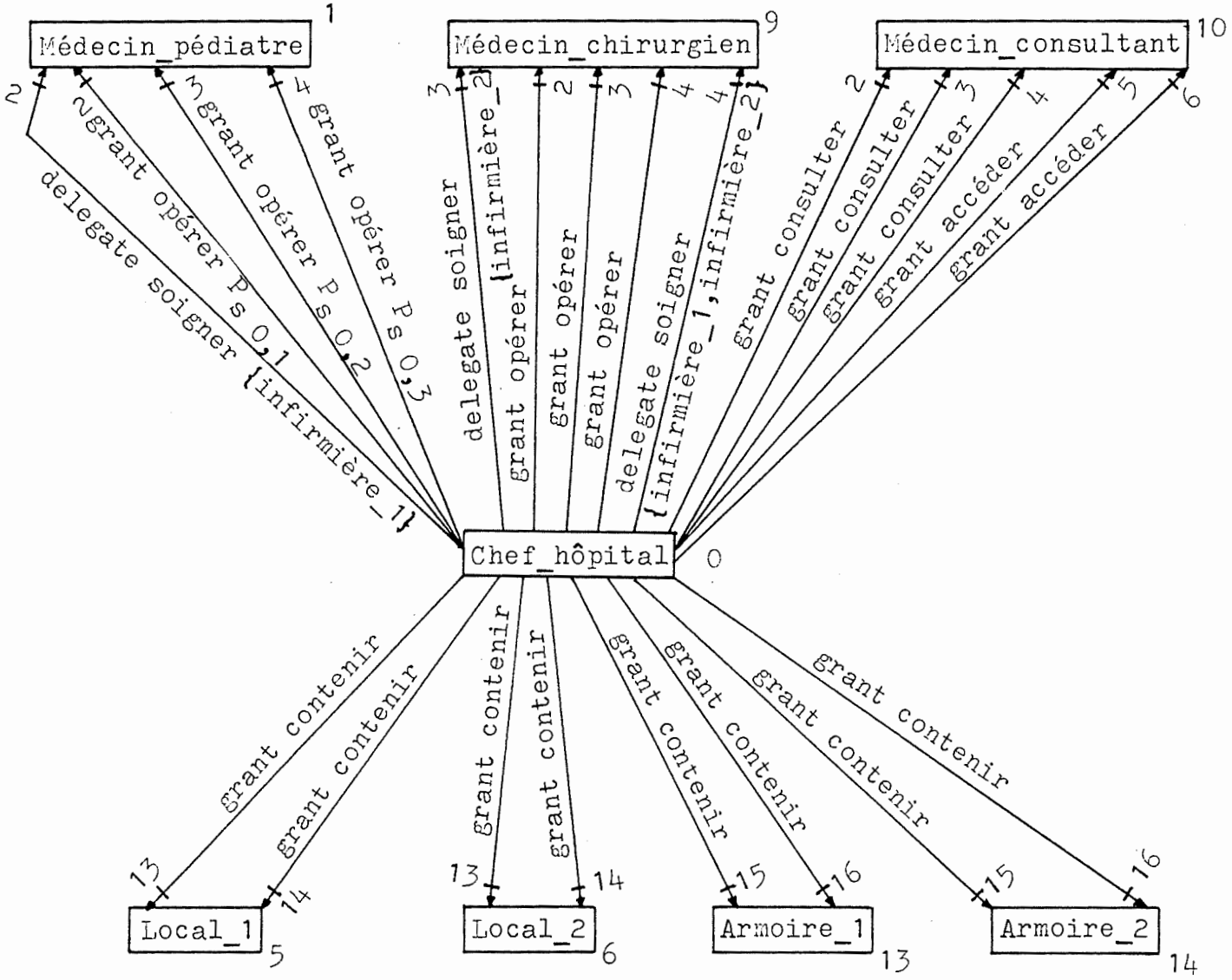
$\{P_{12,7}\}$ = si $\nexists$ x,y : chambre-1 peut contenir x et x est le lit de y et y a été opéré le jour même.

$\{P_{12,8}\}$ = si $\nexists x,y$: chambre-2 peut contenir x et x est le lit de y et y a été opéré le jour même.

N.B. On peut trouver ici les trois types de prédicats dont nous avons supposé l'existence au point 2.1.

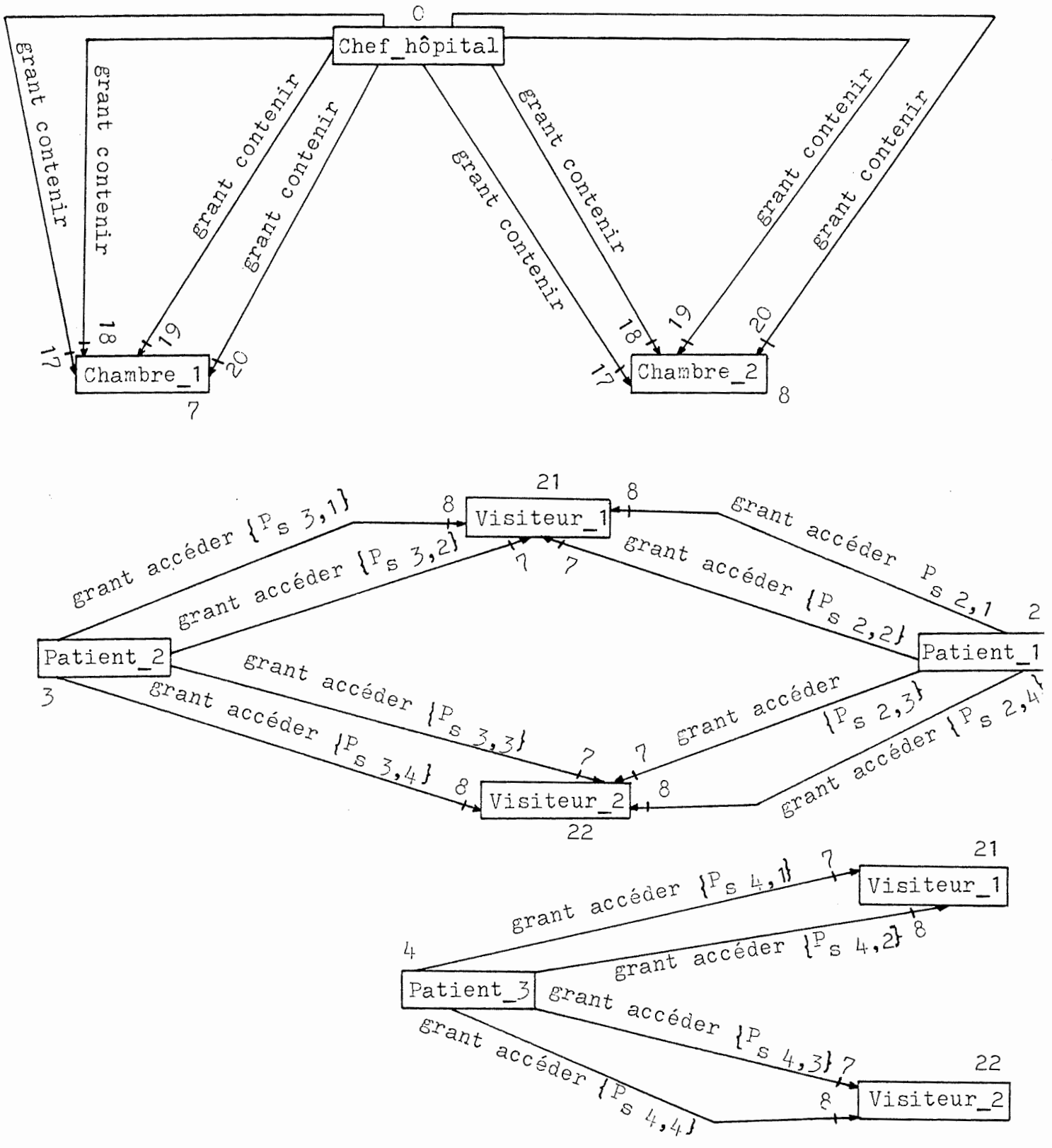
- prédicat portant sur l'existence d'un droit,
- pédricat portant sur les attributs d'une entité,
- prédicat portant sur l'utilisation effective d'un droit.

b. Expression du graphe dynamique DG.



N.B. 15 = Médicaments-1

16 = Médicaments-2



N.B. 17 = Lit-1; 18 = Lit-2: 19 = Lit-3: 20 = Lit-4.

Expression des contraintes.

$\{P_{s\ 0,1}\}$ = si patient-1 est un enfant et
si médecin-chirurgien use effectivement de son droit d'opérer patient-1.

$\{P_{s\ 0,2}\}$ = si patient-2 est un enfant et
si médecin-chirugien use effectivement de son droit d'opérer patient-2.

$\{P_{s\ 0,3}\}$ = si patient-3 est un enfant et
si médecin-chirurgien use effectivemet de son droit d'opérer patient-3.

$\{P_{s\ 2,1}\}$ = si $\exists x$: chambre-1 peut contenir x et
x est le lit de patient-1.

$\{P_{s\ 2,2}\}$ = si $\exists x$: chambre-2 peut contenir x et
x est le lit de patient-1.

$\{P_{s\ 3,1}\}$ = si $\exists x$: chambre-1 peut contenir x et
x est le lit de patient-2.

$\{P_{s\ 3,2}\}$ = si $\exists x$: chambre-2 peut contenir x et
x est le lit de patient-2.

$\{P_{s\ 4,1}\}$ = si $\exists x$: chambre-1 peut contenir x et
x est le lit de patient-3.

$\{P_{s\ 4,2}\}$ = si $\exists x$: chambre-2 peut contenir x et
x est le lit de patient-3.

$\{P_{s\ 2,3}\} = \{P_{s\ 2,1}\}$

$\{P_{s\ 2,4}\} = \{P_{s\ 2,2}\}$

$\{P_{s\ 3,3}\} = \{P_{s\ 3,1}\}$ $\quad\quad$ $\{P_{s\ 4,3}\} = \{P_{s\ 4,1}\}$

$\{P_{s\ 3,4}\} = \{P_{s\ 3,2}\}$ $\quad\quad$ $\{P_{s\ 4,4}\} = \{P_{s\ 4,2}\}$

c. Table de hiérarchisation.

Actions statiques :

opérer
soigner
consulter
nettoyer
accéder
contenir

Actions dynamiques :

delegate/abrogate
grant/revoke

d. Table de classes de sécurité.

Entité	Action maxi. exécutable	Action max. l'entité peut subir
Chef-hôpital	accéder	---
Médecin-pédiatre	opérer	---
Médecin-chirurgien	opérer	---
Médecin-consultant	soigner	---
Infirmière-1	soigner	---

Infirmière-2	soigner	---
Infirmière-3	soigner	---
Infirmière-4	soigner	---
Femme-ouvrage	nettoyer	---
Patient-1	accéder	opérer
Patient-2	accéder	opérer
Patient-3	accéder	soigner
Local-1	contenir	nettoyer
Local-2	contenir	nettoyer
Armoire-1	contenir	contenir
Armoire-2	contenir	contenir
Médicament-1	---	contenir
Médicament-2	---	contenir
Visiteur-1	accéder	---
Visiteur-2	accéder	---
Chambre-1	contenir	nettoyer
Chambre-2	contenir	nettoyer
Lit-1	---	contenir
Lit-2	---	contenir

Lit-3	---	contenir
Lit-4	---	contenir

e. Table de possession d'entités.

Entité propriétaire	Entités possédées
Chef-hôpital	patient-1, patient-2. patient-3
Chef-hôpital	chambre-1, chambre-2, local-1
Chef-hôpital	local-2. armoire-1, armoire-2
Chef-hôpital	lit-1, lit-2, lit-3. lit-4

f. Contrôle de la modélisation obtenue par les règles de cohérence.

Règle de cohérence interne du SG (cfr. point 3.1.3.1.)

La règle de cohérence interne du graphe statique SG a été présentée au point 3.1.3.1. à la page 36; elle nous dit qu'une entité ne peut pas être autorisée à exécuter ou à subir une action incompatible avec sa nature et son rôle dans le système (tables de classes de sécurité).

Le graphe statique de notre modélisation respecte cette règle car aucune entité ne dispose d'action statique de niveau hiérarchique supérieur à sa potentialité active, aucune entité ne subit d'action de niveau hiérarchique supérieur à sa potentialité passive.

Règles de cohérence interne du DG (cfr point 3.1.3.2.)

Les règles de cohérence interne du graphe dynamique DG ont été présentées au point 3.1.3.2. à la page 41. Reprenons les afin de montrer que notre modélisation les respecte :

Règle I1 : Soit E_j l'entité propriétaire d'une entité E_k. Aucune autre entité E_i ($i \neq j \neq k$) ne peut exécuter une action $A_{ij/k}$ de type GRANT/REVOKE SA où E_k est l'entité-paramètre.

Cette règle est bien vérifiée dans notre modélisation car aucune entité ne possède d'action dynamique sur "Chef-hôpital" qui est le seul possesseur.

Règle I2 : Seule l'entité E_i propriétaire d'une entité E_k peut exécuter une action $A_{ij/k}$ de type DELEGATE/ABROGATE.

Les seules actions DELEGATE/ABROGATE de notre modélisation sont celles où le "Chef-hôpital" délègue ses pouvoirs sur des entités qu'il possède ("Patient-1", "Patient-2" et "Patient-3").

Règle I3 : Soit une entité E_j autorisée à gratifier à une entité E_m ($m \neq i$) une SA de niveau L' avec paramètre-entité E_k. L'entité E_i propriétaire de E_k ne peut exécuter des actions $A_{ij/k}$ de type DELEGATE SA que si $L(SA) > L'$.

Les seules entités pouvant subir un DELEGATE/ABROGATE sont "Médecin-pédiatre" et "Médecin-chirurgien"; aucune de ces entités ne dispose d'action de GRANT/REVOKE: il s'ensuit que la règle I3 est vérifiée dans notre modélisation.

Règle I4 : Une entité E_j qui s'est vue déléguée une SA de niveau L ne peut gratifier cette SA qu'aux entités E_m dont a_m^+, ce qui constitue une condition sur l'ensemble $\{M\}$ de l'action DELEGATE/ABROGATE.

Toutes les délégations de notre modèle se font vers les

infirmières pour une opération statique "soigner"; étant donné que la potentialité active de toutes ces infirmières est "soigner", notre modélisation respectera sans conteste cette règle I4.

Règles de cohérence mutuelle entre le SG et le DG (cfr. point 3.1.3.2.)

Les règles de cohérence mutuelle entre le graphe statique SG et le graphe dynamique DG ont été présentées au point 3.1.3.2. à la page 42. Reprenons les afin de voir si notre modélisation les respecte :

Règle M1 : Une entité E_i ne peut gratifier à une autre entité E_j le droit d'exécuter une SA de niveau L avec paramètre-entité E_k que si, dans le SG, E_i peut exécuter une SA de niveau $L' \geq L$ sur E_k.

Notre modélisation ne respecte pas cette règle car l'entité "Chef-hôpital" gratifie des actions statiques qu'elle ne possède pas dans le graphe statique (par exemple, "chef-hôpital" gratifie au "Médecin-chirurgien" l'opération "opérer" sur l'entité "Patient-1" alors que ce "Chef-hôpital" ne possède pas dans le SG d'action de niveau hiérarchique supérieur ou égal à "opérer" sur "Patient-1").

Règle M2 : Si à l'initialisation du système dans le SG, une entité E_j peut exécuter une SA A_{jk} de niveau L, alors, dans le DG, une entité E_i ne peut gratifier à E_j le droit d'exécuter une action statique SA que si $L(SA) > L$.

Notre modélisation présente, comme nous l'avons dit, une structure hospitalière à un moment donné de son existence : la règle M2 porte sur le début de la vie de la structure si bien que pour respecter M2, il faut ajouter à notre modélisation l'hypothèse qu'au début de l'existence de la structure les entités "Médecin-chirurgien", "Médecin-pédiatre" et "Médecin-consultant" n'ont aucun droit d'action sur "Patient-1", "Patient-2" ou "Patient-3".

Règle M3 : A toute action $A_{ij/k}$ de type DELEGATE SA du DG doit correspondre, dans le SG, une action A_{ij} de type CREATE/DELETE. En fait, l'action DELEGATE SA avec entité-paramètre E_k ne peut être exécutée que par le propriétaire E_i de E_k et E_i doit donc être nécessairement autorisée à exécuter l'action CREATE/DELETE sur E_k.

Notre modélisation ne respecte pas cette règle car l'entité "Chef-hôpital" possède des entités sur lesquelles il n'a pas le droit d'exécuter une action CREATE/DELETE.

3.4.4. Conclusions sur la modélisation.

Dans notre structure hospitalière, le chef-hôpital correspond à une entité qui coordonne la structure en déléguant et en gratifiant des droits qu'il ne possède pas lui-même. Dans le modèle ACTEN, il n'est pas possible de représenter celà.

Problème des prédicats : peut-on exprimer dans le modèle les trois types de contraintes rencontrés dans la modélisation ?

L'utilisation de ce modèle implique que l'on sache dès le départ quelles sont les entités que l'on peut créer, quels sont les droits qu'une entité possèdera sur les entités qui seront créées et quels seront les droits de ces entités nouvellement créées; ceci limite étroitemet l'évolution possible de la structure modélisée. Dans ce modèle, on peut donc dire qu'une entité a le droit de créer une entité précisée à l'avance, mais n'a pas le droit de créer une entité quelconque. (contrairement au modèle Take - Grant où là une entité peut créer une entité quelconque).

Dans le chapitre 4, nous tenterons de résoudre ces problèmes en étendant le modèle ACTEN.

3.5. Le problème de sécurité des systèmes d'autorisations issus du modèle ACTEN

Le présent paragraphe a pour objectif de montrer que le problème de sécurité relatif à la quatrième définition de la sécurité, donnée au point 1.5. à la page 13, est décidable et même que ces systèmes sont sûrs.

Rappelons tout d'abord quelle était cette définition de la sécurité : "un système d'autorisatins offre la sécurité si et seulement si le fait qu'un sujet puisse acquérir un droit d'accès sur une informatin résulte directement ou indirectement d'une ou plusieurs actions entreprises par le possesseur de l'information et est connu de celui-ci".

Le problème de sécurité revient donc à répondre à la question suivante : "peut-on, pour toute configuration d'un système d'autorisations issu du modèle ACTEN, déterminer si ou ou non le fait qu un sujet puisse acquérir un droit d'accès sur une information résulte directement ou indirectement d'une ou plusieurs actions entreprises par le possesseur et est connu de celui-ci ?

L'acquisition d'un droit d'accès sur une information ou entité ne peut se faire dans le modèle ACTEN que par une procédure de transformation des graphes statique ou dynamique résultant de l'exécution d'une action de types DELEGATE ou GRANT, ou de l'exécution d'un droit indirect de type GRANT.

Si l'action transformant le graphe dynamique, c'est-à-dire l'action de type DELEGATE, est exécutée, les règles de cohérence interne du graphe dynamique (spécialement la règle I2, énoncée au point 3.1.3.2.) assurent que seule l'entité possédant l'entité-paramètre a les moyens d'exécuter cette action, ce qui permet de répondre "oui" à la question énoncée ci-dessus. Si l'action transformant le graphe statique, c'est-à-dire l'action de type GRANT, est exécutée, l'entité-possesseur n'est pas la seule à pouvoir être à l'origine de l'exécution de cette action. Supposons que l'entité E_k à

l'origine de cette action n'est pas l'entité-possesseur et demandons-nous d'où peut provenir le droit d'exécuter une telle action.

Cette action de type GRANT ne peut provenir que de l'exécution d'une action de type DELEGATE que seule l'entité-possesseur peut exécuter ou de la présence de ce droit d'action à l'initialisation du système.

A condition que l'on convienne de ne donner aucun droit initial sur une entité non possédée, ce qui paraît raisonnable, on peut encore répondre "oui" à la question énoncée ci-dessus.

Si à l'origine d'une acquisition de droit se trouve une action indirecte de type GRANT de E_i à E_j avec paramètre-entité E_k (la règle I2. énoncée au point 3.1.3.2. assure qu'une action indirecte de type DELEGATE ne peut exister), cette action indirecte se trouve nécessairement sur une chaîne de GRANT comprenant une action directe de type GRANT de E1 sur E_j avec paramètre-entité E_k où l'action statique gratifiée est de niveau hiérarchique supérieur ou égal à l'action statique indirecte gratifiée par E_i à E_1 si bien que d'un point de vue possesseur la détermination de ce droit indirect de gratification n'importe pas (le possesseur ne se soucie pas de savoir qui gratifie mais bien qui reçoit).

On peut dès lors toujours répondre "oui" à la question énoncée ci-dessus. Puisque toute acquisition de droits résulte directement ou indirectement d'une ou plusieurs actions entreprises par le possesseur, répondre "oui" au fait que le possesseur sait qu'une entité bien précise peut acquérir un droit bien précis sur l'entité qu'il possède revient à affirmer que le système d'autorisations a les moyens de prévenir le possesseur de la portée des actions qu'il veut entreprendre.

Le moyen de prévenir le possesseur d'une information de la portée des actions qu'il entreprend ou veut entreprendre est offert par l'analyse des états d'autorisation et de protection donnée au point 3.1.4.1. à la page 47. En effet, si le possesseur d'une entité exécute ou signale au système qu'il voudrait exécuter une action, le système pourra exécuter cet algorithme afin de déterminer notamment l'état de protection maximum de l'entité en question: c'est alors au possesseur que revient la possibilité d'annihiler l'effet de cette action ou de ne pas l'exécuter.

CHAPITRE IV

EXTENSION DU MODELE ACTEN

Ce chapitre d'extension du modèle ACTEN donne suite au paragraphe 3.4. où l'utilisation du modèle est étudiée et répond notamment aux problèmes soulevés et repris dans la conclusion de modélisation du point 3.4.4.

Le présent chapitre assurant en quelque sorte l'interface entre un paragraphe 3.4. ouvert sur le monde réel et ses structures et un chapitre 5 où il est exclusivement question de système informatique, les problèmes soulevés au point 3.4.4. risquent de trouver une solution triviale lorsqu'on ne considère que le monde informatique.

Le chapitre 4 traite du problème de gratification de droit non possédé soulevé dans le chapitre 3 du problème de la création dynamique d'entités et de ses implications, du problème de la détermination des potentialités active et passive des entités, du problème de la suppression d'entités du problème de la révocation des droits et enfin de la notion de hiérarchisation des actions.

4.1. GRATIFICATION D'UN DROIT NON POSSEDE.

Dans la modélisation d'une structure hospitalière étudiée au point 3.4. nous avons senti le besoin de permettre à certaines entités, le chef d'hôpital en l'occurence, de disposer d'un droit de gratification d'une action qu'il ne possède pas (le chef-hôpital peut gratifier le droit de soigner le patient-I alors qu'il n'a lui-même pas le droit de soigner ce patient), ce qui est contraire aux règles de cohérence (règle MI de cohérence mutuelle énoncée au point 3.1.3.2.).

On peut donner une solution immédiate à ce problème en disant que le chef-hôpital possède tous les droits sur toutes les entités mais que ces droits sont soumis à des prédicats toujours faux si bien que la règle MI de cohérence mutuelle est vérifiée alors qu'en pratique il ne peut pas exercer ces droits.

Néanmoins, cela signifie qu'il existe dans le système d'autorisations une entité prédéfinie qui a tous les droits sur toutes les autres entités ce

qui est contraire à l'hypothèse du point 3.5.2. grâce à laquelle on a montré la sécurité de systèmes d'autorisations issus du modèle ACTEN.

Dans un système informatique, il existe également une entité qui possède tous les droits sur toutes les autres entités, droits qui sont ici effectifs, et qui est ce que l'on appelle le superuser ou utilisateur privilégié. Cette entité privilégiée possède tous les droits en ce sens qu'elle possède toutes les entités du système d'autorisations et n'est sujette à aucun droit d'action.

Il s'ensuit que le problème de sécurité tel que discuté au point 3.5. reste décidable mais la réponse devient "non" (un tel système n'est pas sûr). Le problème en découlant est que le propriétaire d'une entité ne sait plus contrôler tous les droits possédés sur son entité et ainsi empêcher une action indésirable faite par une entité indésirable.

Il faut donc admettre la non sécurité du système d'autorisation tout en se disant que cet utilisateur privilégié sait ce qu'il fait, sait à qui il gratifie des droits et agit "pour le bien" de l'ensemble des entités en toute connaissance de cause.

4.2. IMPLICATION DE LA CREATION D'ENTITES (création dynamique)

4.2.1. LE PROBLEME.

Tel que le modèle ACTEN nous est présenté, la création d'entités implique, par le jeu des droits indirects la perte de toute sélectivité dans la gratification de droits. En effet, on peut mettre en évidence un mécanisme de gratification de droits échappant au contrôle du graphe dynamique, si bien que la possession d'un droit permet automatiquement son transfert (gratification) à quiconque.

Pour plus de clarté, nous illustrerons ce mécanisme sur un exemple :

a) Considérons une entité E_1 possédant un droit X sur une entité E2

possédée par une entité E_3 : on peut affirmer que la potentialité active de E_1 est au moins X car, dans le cas contraire, les règles de cohérence jouant, le droit X n'aurait pas pu être inséré entre E_1 et E_2.
Considérons d'autre part une entité E n'ayant aucun droit sur E_2 mais ayant également une potentialité active d'au moins X.
Pour la même raison qu celle évoquée pour la potentialité active de E_1, on peut affirmer que la potentialité passive de E_2 est d'au moins X (cfr figure 4-1).

Graphe statique

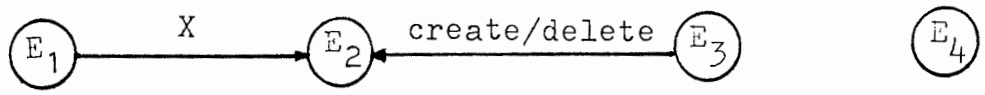


Table de classes de sécurité

Entité	Pot.act.	Pot.pass.
E_1	$\geq$ X	
E_2		$\geq$ X
E_3	create	
E_4	$\geq$ X	

Table d'états d'entités

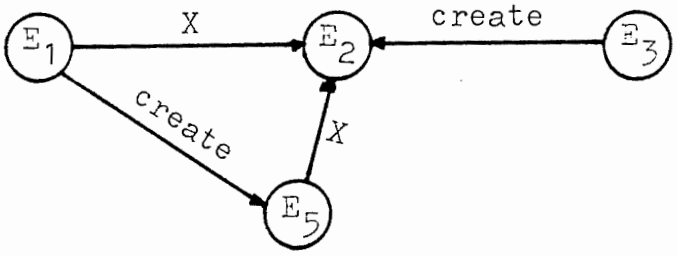
Possesseur	Possédée
E_3	E_2

n.b. : - E_3 a au moins la potentialité active de créer pour pouvoir posséder E_2.
- les entrées non spécifiées dans la table (telle la potentialité passive de E_1) sont indifférentes ici.

Figure 4-1

b) Ajoutons l'hypothèse que non seulement la potentialité active de E_1 est au moins X (comme dit en a) mais de plus que cette potentialité active est create.
Supposons que E_1 crée une entité E_5 sur laquelle E_1, puisque possesseur. acquiert le droit statique le plus élevé (create) et le droit dynamique le plus élevé (delegate) avec E_5 comme entité-paramètre (E_1 peut déléguer un droit statique quelconque sur E_5). Puisque E_5 a été créée par E_1 et est donc destinée à oeuvrer pour E_1, elle doit acquérir tous les droits statiques de E_1 dans la mesure où ces droits ne dépassent pas la potentialité active de E_5.
Supposons que la potentialité active de E_5 est au moins X et que sa potentialité passive est au moins use (utiliser, exécuter) (cfr figure 4-2).

Graphe statique



Graphe dynamique

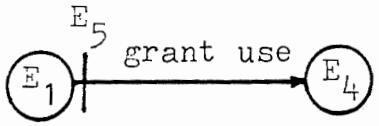


Table de classes de sécurité

Entité	Pot.act.	Pot.pass.
E_1	create	
E_2		≥ X
E_3	create	
E_4	≥ X	
E_5	≥ X	≥ use

Table d'états d'entités

Possesseur	Possédée
E_3	E_2
E_1	E_5

Figure 4-2

c) Supposons que E_1 exécute son droit de grant use sur E_5 à E_4, droit qui existe puisque. comme nous l'avons vu en b-, E_1 possède E_5 et a donc le droit delegate create (droit dyn. le plus élevé) on E_5 vers quiconque. Il s'ensuit que E_4 acquiert le droit use sur E_5. Par le jeu des droits indirects, E_4 a aussi acquis le droit X sur E_2, puisque la potentialité active de E_4 est au moins X. (cfr figure 4-3).

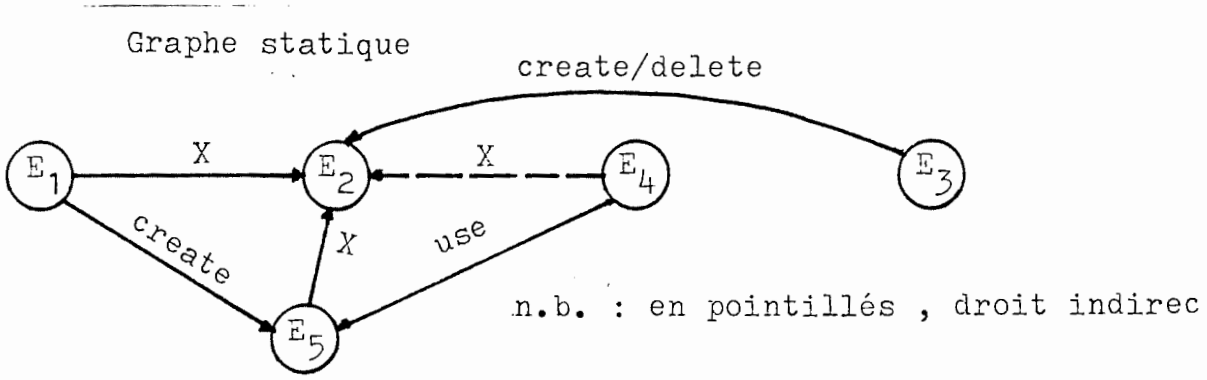


Figure 4-3

d) Voilà donc un mécanisme autorisé par ACTEN ayant permis le transfert du droit X sur E_2 de E_1 vers E_4 sans se soucier si le graphe dynamique présentait ou non un droit "grant X on E_2 from E_1 to E_4".
Si donc ce droit de grant n'existait pas, la gratification s'est tout de même opérée.

4.2.2. LA SOLUTION ENVISAGEE.

La solution envisagée doit avoir un effet sur la détermination du droit indirect uniquement car il faut laisser à E_1, si l'on se réfère à nouveau à la figure 4-3, le droit de gratifier des droits statiques sur son entité E_5 et laisser à E_5 la possibilité d'acquérir les droits statiques de E_1.

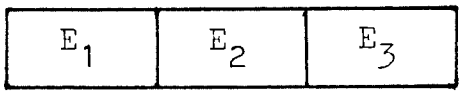


Figure 4-5

3) - le programme d'application E_3 demande pour écrire "write" dans la zone de données E_4.

- le système d'autorisations doit maintenant vérifier si E_1, E_2 et E_3 ont ce droit de "write" sur E_4 et si l'une de ces trois entités ne le possède pas. le système répond : "refusé". (cfr figure 4-6).

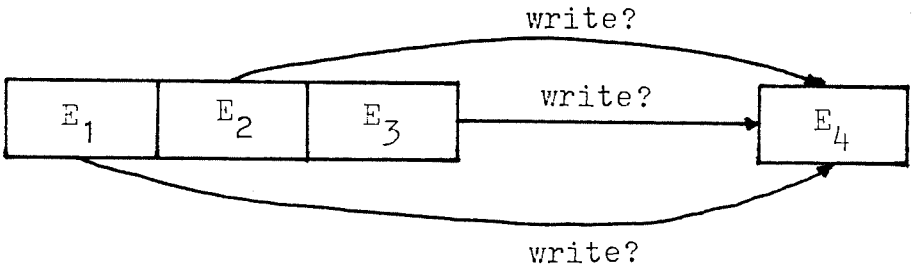


Figure 4-6

QUELLES SONT LES IMPLICATIONS APPORTEES PAR CETTE SOLUTION ?

Il faut que le système d'exploitation mette à la disposition du système d'autorisations des outils permettant de déterminer, lors de toute demande d'exécution d'un droit, qui est l'auteur de la demande.
D'autre part, le système d'autorisations doit oublier, dans le cadre d'un système d'exploitation multi-tasking, la notion de programme stricte mais l'étendre à la notion de processus; exécuter un programme revient alors à 1° - créer un processus et à 2° - exécuter ce processus.
Sans celà, le système d'autorisations est incapable de déterminer, lors d'une demande d'exécution d'un droit, à quelle chaîne d'utilisation se rattache cette demande, à moins que le système d'exploitation soit mono-tasking.

4.3. LA DETERMINATION DES POTENTIALITES ACTIVE ET PASSIVE DES ENTITES.

4.3.1. LE PROBLEME.

Lors de la création d'une entité, qui va déterminer les potentialités active et passive de cette nouvelle entité et comment le fera-t-il ?

4.3.2. LA SOLUTION ENVISAGEE.

Le rôle joué par la potentialité dans le modèle ACTEN est de restreindre les acquisitions de droits en fonction de la mission attribuée aux entités du système, c'est-à-dire en fonction de ce qu'elles sont sensées devoir faire.

Or, qui mieux que l'entité créatrice peut connaître la mission de l'entité qu'elle crée ?

L'entité créatrice crée une entité devant oeuvrer pour elle et lui attribue donc une mission, une tâche (un utilisateur par exemple crée un programme qui a pour mission de l'aider à résoudre un problème); il s'ensuit que l'entité créatrice devrait avoir son mot à dire pour l'attribution des potentialités de l'entité qu'elle crée.

D'autre part, on imagine difficilement la création d'un outil, d'une entité, qui aurait une mission plus vaste que celle attribuée à l'entité créatrice; il s'ensuit que l'attribution des potentialités de l'entité créée doit être soumise à majoration par les potentialités de l'entité créatrice.

En synthèse, la solution envisagée est de laisser à l'entité créatrice le choix des potentialités de l'entité créée dans la mesure où elles ne dépassent pas ses propres potentialités.

4.4. LA SUPPRESSION (delete) D'ENTITES.

4.4.1. LE PROBLEME.

Le problème est le suivant : lorsque l'on supprime une entité E_i du système d'autorisations que doit-on faire des entités possédées par E_i?

4.4.2. LA SOLUTION ENVISAGEE.

S'il semble "normal" de supprimer toutes les entités possédées par un utilisateur lorsque cet utilisateur est rayé du système d'exploitation et donc du système d'autorisations, il n'en va pas de même pour un programme d'application, par exemple.

Considérons, par exemple, le cas d'un programme d'application ayant pour mission de créer et remplir une zone de données et qui, après avoir accompli sa mission, est effacé du système pa l'utilisateur qui a créé ce programme, pour des raisons de limitation de mémoire disponible par exemple. Peut-on envisager alors d'effacer également cette zone de données alors que peut-être un autre programme d'application aurait pour mission de mettre à jour cette zone de données ? La réponse est bien entendu "non". Que faire alors de cette zone de données ayant perdu son possesseur et peut-être inutilisable si le créateur/possesseur n'a pas délégué de droit sur elle du temps de son vivant ?

La solution envisagée est d'attribuer au possesseur direct de l'entité E_i que l'on supprime toutes les entités que E_i possédait avant de mourir.
Cette solution implique le fait que si les règles de cohérence "physique" autorise E_i à posséder certains types d'entités (user, programme , zone-données, ...), il doit en aller de même pour le possesseur de E_i ainsi que pour le possesseur du possesseur de E_i, etc,...
Ainsi donc, un type d'entité donné ne peut créer une entité dont le type permettrait "physiquement" d'exercer plus de droits que ne le permet le type de l'entité créatrice (un programme d'application, par exemple, ne peut pas créer un utilisateur).

4.5. PROBLEME DE LA REVOCATION DE DROITS.

Jusqu'à présent, nous n'avons parlé que d'acquisitions de droits et peu de révocations de droits. Or, la révocation de droits peut poser certains problèmes intéressants à étudier :

I. Deux entités se contredisent :

Si comme le montre la figure 4-7, deux entités E_1 et E_2 ont toutes deux le droit dynamique de gratifier une même action statique (même opération et même prédicat) à une entité E_3 sur une entité-paramètre E_4, il se peut que E_I exécute cette action dynamique de Grant afin de donner à E_4 les moyens de travailler pour elle et que E_2 exécute l'action dynamique Revoke associée au Grant, et ce un nombre considérable de fois.

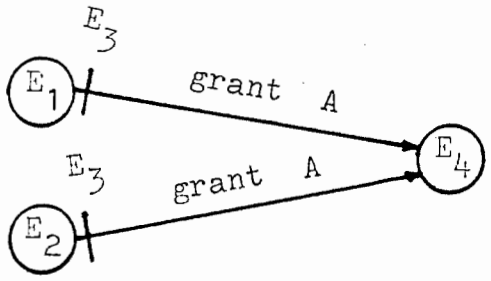


Figure 4-7. Configuration permettant la contradiction entre entités.

Le problème est de savoir si cet alternance de Grant par E_1 et REVOKE par E_2 est un cas pathologique.

Il se peut que ce le soit et que E_2 joue donc le rôle de trouble-fête (ou vice-versa).

Il se peut que cela ne le soit pas: si, par exemple, E_I représente un processus d'allocation d'une ressource telle qu'une variable dynamique et que E_2 représente un processus de désallocation de cette ressource. le cas n'est pas pathologique car E_I alloue une variable

dynamique à E_4 et E_2 la désalloue quand E_4 a terminé son travail avec cette variable. et ce de façon répétitive si E_4 boucle sur une même séquence d'instructions.

Le système d'autorisations ne peut rien faire en lui-même pour déterminer si c'est un cas pathologique mais pourrait avertir le possesseur de E_3 de ce qui se passe et lui laisser la décision.

4.6. LA NOTION DE HIERARCHISATION D'ACTIONS

Nous avons évoqué au point 3.1.2.2. la notion de hiérarchisation des opérations dans le modèle ACTEN afin de minimaliser le nombre d'actions à représenter dans les graphes.

De cette notion de hiérarchisation des opérations à une notion de hiérarchisation d'actions, il n'y a qu'un pas que l'on franchira toujours dans ce but de minimalisation. Une action étant constituée d'une opération et d'un ensemble de prédicats, le pas en question concerne la définition d'une relation entre ensembles de prédicats qui, couplée a la notion de hiérarchisation des opérations, fournira une notion de hiérarchisation des actions.
Nous allons franchir ce pas en quatre étapes ;

1. - définition des types de prédicats retenus pour la suite.
2. - définition d'une relation de contraignance entre deux prédicats.
3. - définition d'une relation de contraignance entre deux ensembles de prédicats.
4. - définition d'une notion de hiérarchisation des actions.

4.6.1. LES DIFFERENTS TYPES DE PREDICATS RETENUS

Les sept types de prédicat retenus pour la suite sont :

A. TYPE TEMPOREL

Le type temporel permet de définir un laps de temps durant lequel

l'opération associée est autorisée. Deux données sont nécessaires :

1 - heure de début
2 - heure de fin

B. TYPE PORTANT SUR L'EXISTENCE D'UNE ACTION DYNAMIQUE SANS PREDICAT

Le type portant sur l'existence d'une action dynamique sans prédicat permet d'exprimer une condition portant sur le contenu du graphe dynamique. Cinq données sont nécessaires :

1 - entité agissante
2 - entité subissante
3 - opération dynamique
4 - entité paramètre
5 - opération statique

C. TYPE PORTANT SUR L'EXISTENCE D'UNE ACTION STATIQUE SANS PREDICAT

Le type portant sur l'existence d'une action statique sans prédicat permet d'exprimer une condition portant sur le contenu du graphe statique.

Trois données sont nécessaires :

1 - entité agissante
2 - entité subissante
3 - opération statique

D. TYPE PORTANT SUR LA NATURE PHYSIQUE D'UNE ENTITE

A ce stade,nous devons définir les natures physiques utilisées ultérieurement (cfr implémentation) ainsi que hiérarchiser ces natures· nous retiendrons trois natures physiques que nous citons par ordre hiérarchique décroissant : user ou utilisateur, programme, zone de données.

Le type portant sur la nature physique d'une entité permet d'exprimer une condition telle que "si E_i est au moins de nature physique X".
Deux données sont nécessaires :

1 - entité concernée
2 - nature physique minimale

E. TYPE PORTANT SUR LA POTENTIALITE ACTIVE D'UNE ENTITE

Le type portant sur la potentialité active d'une entité permet d'exprimer une condition telle que "si E_i a au moins la potentialité active X" (X étant une opération statique).
Deux données sont nécessaires :

1 - entité concernée
2 - potentialité active minimale

F. TYPE PORTANT SUR LA POTENTIALITE PASSIVE D'UNE ENTITE

Le type portant sur la potentialité passive d'une entité permet d'exprimer une condition telle que "si E_i a au moins la potentialité passive X" (X étant une opération statique).
Deux données sont nécessaires :

1 entité concernée
2 - potentialité passive minimale

G. TYPE PORTANT SUR L'EXISTENCE D'UNE RELATION DE POSSESSION

Le type portant sur l'existence d'une relation de possession permet d'exprimer une condition telle que "si E_i possède E_j".
Deux données sont nécessaires :

1 - entité possédant
2 - entité possédée

4.6.2. LA NOTION DE CONTRAIGNANCE ENTRE DEUX PREDICATS

La notion de contraignance est fonction des types des prédicats et des contraintes qu'ils expriment: cette notion est assez intuitive en ce qui concerne le type de prédicat temporel et fait appel au contexte de hiérarchisation des opérations dans les autres cas (sauf le cas trivial du

type portant sur l'existence d'une relation de possession). Nous donnerons une définition de cette notion au moyen de tables de décision.

Une table de décision est la représentation sous forme de table d'un ensemble de règles de décision, faisant apparaître les correspondances entre conditions et actions ou conclusions; la moitié gauche de la table, que l'on appelle "souche", contient l'énoncé des conditions (1er quadrant) et des activités ou conclusions (4ème quadrant); la moitié droite de la table, que l'on appelle "entrées", contient les valeurs prises par les conditions (2ème quadrant) et les actions à effectuer ou conditions correspondantes (3ème quadrant); une colonne de cette partie droite définit donc une "règle de décision". Dans les tables qui suivent, les valeurs "y" indiquent des conditions vérifiées ou des conclusions vérifiées, les valeurs "n" des conditions non vérifiées ou des conclusions non vérifiées; enfin, le vide signale l'indifférence de la règle à la condition correspondante.

a) - Si P1 et P2 sont deux prédicats de types distincts, on dira qu'ils sont de contraintes différentes ou incomparables.

b) - Si P1 et P2 sont tous deux de type temporel :

CONDITIONS	h_d(P1) h_d(P2)	y	y	y	n	n	n	n	n	n
	h_d(P1) = h_d(P2)	n	n	n	n	n	n	y	y	y
	h_f(P1) h_f(P2)	n	n	y	n	y	n	y	n	n
	h_f(P1) = h_f(P2)	y	n	n	y	n	n	n	n	y
CONCLUSIONS	P1 est moins contraignant que P2	y	y	n	n	n	n	n	y	n
	P1 est plus contraignant que P2	n	n	n	y	y	n	y	n	n
	P1 est de contrainte égale à P2	n	n	n	n	n	n	n	n	y
	P1 est de contrainte différente de P2	n	n	y	n	n	y	n	n	n

Table 1 : Notion de contraignance entre deux prédicats de type temporel

c) - Si P1 et P2 sont tous deux de type portant sur l'existence d'une action dynamique sans prédicat :

CONDITIONS	ent_ag(P1) = ent_ag(P2)	n	y	y	y	y	y	y	y	y	y	y	y
	ent_sub(P1) = ent_sub(P2)		n	y	y	y	y	y	y	y	y	y	y
	ent_par(P1) = ent_par(P2)			n	y	y	y	y	y	y	y	y	y
	op_d(P1) n.h.i. op_d(P2)				y	y	y	n	n	n	n	n	n
	op_d(P1) n.h.é. op_d(P2)				n	n	n	n	n	n	y	y	y
	op_s(P1) n.h.i. op_s(P2)				y	n	n	n	n	y	y	n	n
	op_s(P1) n.h.é. op_s(P2)				n	y	n	n	y	n	n	n	y
CONCLUSIONS	P1 est moins contraignant que P2	n	n	n	y	y	n	n	n	n	y	n	n
	P1 est plus contraignant que P2	n	n	n	n	n	n	y	y	n	n	y	n
	P1 est de contrainte égale à P2	n	n	n	n	n	n	n	n	n	n	n	y
	P1 est de contrainte différente de P2	y	y	y	n	n	y	n	n	y	n	n	n

Table 2 : Notion de contraignange entre deux prédicats de type portant sur l'existence d'une action dynamique sans prédicat.

d) - Si P1 et P2 sont tous deux de type portant sur l'existence d'une action statique sans prédicat :

CONDITIONS	ent_ag(P1) = ent_ag(P2)	n	y	y	y	y
	ent_sub(P1) = ent_sub(P2)		n	y	y	y
	op_s(P1) n.h.i op_s(P2)			y	n	n
	op_s(P1) n.h.é op_s(P2)			n	y	n
CONCLUSIONS	P1 est moins contraignant que P2	n	n	y	n	n
	P1 est plus contraignant que P2	n	n	n	n	y
	P1 est de contrainte égale à P2	n	n	n	y	n
	P1 est de contrainte différente de P2	y	y	n	n	n

Table 3 : Notion de contraignance entre deux prédicats de type portant sur l'existence d'une action statique sans prédicat.

e) - Si P1 et P2 sont tous deux de types portant sur la nature physique d'une entité :

CONDITIONS	ent_conc(P1) = ent_conc(P2)	n	y	y	y
	nat_phys(P1) n.h.i. nat_phys(P2)		y	n	n
	nat_phys(P1) n.h.é. nat_phys(P2)		n	y	n
CONCLUSIONS	P1 est moins contraignant que P2	n	y	n	n
	P1 est plus contraignant que P2	n	n	n	y
	P1 est de contrainte égale à P2	n	n	y	n
	P1 est de contrainte différente de P2	y	n	n	n

Table 4 : Notion de contraignance entre deux prédicats de type portant sur la nature physique d'une entité.

f) - Si P1 et P2 sont tous deux de type portant sur la potentialité active d'une entité :

CONDITIONS	ent_conc(P1) = ent_conc(P2)	n	y	y	y
	pot_act(P1) n.h.i. pot_act(P2)		y	n	n
	pot_act(P1) n.h.é. pot_act(P2)		n	y	n
CONCLUSIONS	P1 est moins contraignant que P2	n	y	n	n
	P1 est plus contraignant que P2	n	n	n	y
	P1 est de contrainte égale à P2	n	n	y	n
	P1 est de contrainte différente de P2	y	n	n	n

Table 5 : Notion de contraignance entre deux prédicats de type portant sur la potentialité active d'une entité.

g) - Si P1 et P2 sont tous deux de type portant sur la potentialité passive d'une entité :

CONDITIONS	ent_conc(P1) = ent_conc(P2)	n	y	y	y
	pot_pass(P1) n.h.i. pot_pass(P2)		y	n	n
	pot_pass(P1) n.h.é. pot_pass(P2)		n	y	n
CONCLUSIONS	P1 est moins contraignant que P2	n	y	n	n
	P1 est plus contraignant que P2	n	n	n	y
	P1 est de contrainte égale à P2	n	n	y	n
	P1 est de contrainte différente de P2	y	n	n	n

Table 6 : Notion de contraignance entre deux prédicats de type portant sur la potentialité passive d'une entité.

h) - Si P1 et P2 sont tous deux de type portant sur l'existence d'une relation de possession :

CONDIT.	ent_possesseur(P1) = ent_possesseur(P2)	n	y	y
	ent_possédée(P1) = ent_possédée(P2)		n	y
CONCL.	P1 est de contrainte égale à P2	n	n	y
	P1 est de contrainte différente de P2	y	y	n

Table 7 : Notion de contraignance entre deux prédicats de type portant sur l'existence d'une relation de possession entre deux entités.

4.6.3. LA NOTION DE CONTRAIGNANCE ENTRE DEUX ENSEMBLES DE PREDICATS

On dira qu'un ensemble de prédicats Ep1 est moins contraignant qu'un autre ensemble de prédicats Ep2 si et seulement si :

1. - pour tout prédicat de Ep1, il existe au moins un prédicat de Ep2 plus contraignant.

2. - il existe au moins un prédicat de Ep1 vis-à-vis duquel aucun prédicat de Ep2 n'est moins contraignant.

On dira qu'un ensemble de prédicats EP1 est plus contraignant qu'un autre ensemble de prédicats Ep2 si et seulement si :

1. - pour tout prédicat de Ep1, il existe au moins un prédicat de Ep2

moins contraignant.

2. -il existe au moins un prédicat de Ep1 vis-à-vis duquel aucun prédicat de Ep2 n'est plus contraignant.

On dira qu'un ensemble de prédicats Ep1 est de contrainte différente d'un ensemble de prédicats Ep2 si et seulement si

1. - il existe au moins un prédicat de Ep1 vis-à-vis duquel aucun prédicat de Ep2 n'est plus contraignant.

2. - il existe au moins un prédicat de Ep1 vis-à-vis duquel aucun prédicat de Ep2 n'est moins contraignant.

3. - il existe au moins un prédicat de Ep1 vis-à-vis duquel aucun prédicat de Ep2 n'est également contraignant.

On dira qu'un ensemble de prédicats Ep1 est également contraignant qu'un ensemble de prédicats Ep2 si et seulement si

1. - Ep1 n'est pas moins contraignant que Ep2
2. - Ep1 n'est pas plus contraignant que Ep2
3. - Ep1 n'est pas de contrainte différente de Ep2

4.6.4. LA NOTION DE HIERARCHISATION DES ACTIONS

4.6.4.1. HIERARCHISATION DES ACTIONS DYNAMIQUES

La notion de hiérarchisation des actions dynamiques est définie dans la table 8 (table de décision) où l'on établit une relation hiérarchique entre deux actions $A_1 \sim$ op-d_1 act -s_1 Ep_1 et $A_2 \sim$ op-O_2 act -s_2 Ep_2.

CONDITIONS	op_d1 n.h.s. op_d2	y	y	y	n	n	n	n	n	n	n	n	n	n
	op_d1 n.h.i. op_d2	n	n	n	y	y	y	n	n	n	n	n	n	n
	op_d1 n.h.é. op_d2	n	n	n	n	n	n	y	y	y	y	y	y	y
	act_s1 n.h.i. act_s2	n	n	y			n	y	y	n	n	n	n	n
	act_s1 n.h.é. act_s2			n			n	n	n	n	n	y	y	y
	act_s1 n.h.s. act_s2			n	n	n	y	n	n	y	y	n	n	n
	Ep1 est plus contraignant que Ep2	n				n			n	n		y	n	n
	Ep1 est moins contraignant que Ep2		n		n			n			n	n	y	n
	Ep1 est de contrainte égale à Ep2		n			n			n		n	n	n	y
	Ep1 est de contrainte différente de Ep2	n			n			n		n		n	n	n
CONCLUSIONS	A1 n.h.i. A2	n	n	n	y	n	n	y	n	n	n	y	n	n
	A1 n.h.é. A2	n	n	n	n	n	n	n	n	n	n	n	y	n
	A1 n.h.s. A2	y	n	n	n	n	n	n	n	y	n	n	y	n
	A1 n.h.inc. A2	n	y	y	n	y	y	n	y	n	y	n	n	n

Table 8 : Notion de hiérarchisation des actions dynamiques

4.6.4.2. HIERARCHISATION DES ACTIONS STATIQUES

La notion de hiérarchisation des actions statiques est définie dans la table 9 (table de décision) où l'on établit une relation hiérarchique entre deux actions $A_1 \sim$ op-s_1 Ep_1 et $A_2 \sim$ op-s_2 Ep_2.

CONDITIONS	op_s1 n.h.i. op_s2	y	y	n	n	n	n	n	n
	op_s1 n.h.é. op_s2	n	n	n	n	y	y	y	y
	op_s1 n.h.s. op_s2	n	n	y	y	n	n	n	n
	Ep1 est moins contraignant que Ep2	n			n	y	n	n	n
	Ep1 est plus contraignant que Ep2		n	n		n	y	n	n
	Ep1 est de contrainte égale à Ep2		n		n	n	n	y	n
	Ep1 est de contrainte différente de Ep2	n		n		n	n	n	y
CONCLUSIONS	A1 n.h.i. A2	y	n	n	n	n	y	n	n
	A1 n.h.é. A2	n	n	n	n	n	n	y	n
	A1 n.h.s. A2	n	n	y	n	y	n	n	n
	A1 n.h.inc. A2	n	y	n	y	n	n	n	y

Table 9 : Notion de hiérarchisation des actions statiques

C H A P I T R E V

IMPLEMENTATION D'UN SYSTEME D'AUTORISATIONS ISSU DU MODELE A C T E N

Le chapitre 4 a déjà quelque peu introduit celui-ci en discutant d'extensions ou en précisant certains points du modèle ACTEN. Le chapitre 5 discutera des spécifications du système d'autorisations que l'on veut implémenter, de l'analyse fonctionnelle et de l'architecture logique du logiciel. Ce chapitre discutera donc des traits essentiels du logiciel et laissera au second livre le soin de rentrer plus dans les détails (manuel de l'utilisateur, conception des modules, code source).

5.1. SPECIFICATIONS DU SYSTEME D'AUTORISATIONS

Le système d'autorisations reprend donc les traits essentiels d'ACTEN auxquels viennent s'ajouter les extensions exposées au chapitre 4.
Nous allons ici passer en revue les diverses extensions, précisions et instanciations qui conduisent d'ACTEN au système que l'on désire implémenter.

5.1.1. LES EXTENSIONS PAR RAPPORT AU MODELE A C T E N

Les principales extensions ont été exposées au chapitre 4 : il s'agit de la création "dynamique" d'entités et de la hiérarchisation des actions.

Une extension dont on n'a pas encore parlé se situe au niveau de l'analyse des états d'autorisation et de protection : le système à construire possèdera une application Query qui, en plus de l'analyse des états d'autorisation et de protection, permettra de donner et le chemin d'action et l'ensemble des prédicats associés à toute action issue d'une telle analyse.

5.1.2. LES PRECISIONS PAR RAPPORT AU MODELE A C T E N

Les précisions apportées au modèle ACTEN ont également fait l'objet d'une discussion dans le chapitre 4 : il s'agit de la suppression d'entités et de la détermination des potentialités d'une entité nouvellement créée.

5.1.3. LES INSTANCIATIONS DU MODELE A C T E N

La première instanciation du modèle ACTEN concerne la hiérarchisation des opérations.
La hiérarchisation des opérations dynamiques adoptée est la suivante :

delegate n.h.s. abrogate n.h.s. grant n.h.s. revoke

où n.h.s. signifie "est de niveau hiérarchique supérieur à"

La hiérarchisation des opérations statiques adoptée est la suivante :

create n.h.s. delete n.h.s. write n.h.s. use n.h.s. read

La seconde instanciation concerne la hiérarchisation des diverses natures physiques qu'une entité peut avoir :

user n.h.s. programme n.h.s. zone-données

La troisième instanciation concerne la détermination des droits indirects dans le système à construire. Seule une opération "use" peut donner lieu à un droit indirect (dire qu'une opération "read" donne lieu a un droit indirect n'a à mon avis aucun sens; celà pourrait mieux se comprendre pour une opération "write" mais ne pourrait se concevoir que dans le contexte d'un système d'autorisations plus intelligent).

La quatrième et dernière instanciation concerne les règles de cohérence du système :

a) Règles de cohérence physiques :

Les règles de cohérence physiques expriment certaines limites sur les possibilités de détention d'actions en fonction des natures physiques des entités concernées.
Ces règles sont assez intuitives et expriment les idées suivantes :

- un "user" peut gratifier ou révoquer des actions statiques à un "user" ou à un "programme".
- un "user" peut écrire, lire un "programme" ou une "zone-données".
- un "user" peut exécuter un "programme".
- un "programme" peut gratifier ou révoquer des actions statiques à un autre "programme".
- un "programme" peut créer, effacer, écrire ou lire un autre "programme" ou une "zone-données".
- tout autre cas n'est pas permis.

b) Règles de cohérence relatives aux potentialités :

Il n'est pas permis d'insérer dans le graphe statique une opération de niveau hiérarchique supérieur à la potentialité active de l'entité agissante ou à la potentialité passive de l'entité subissante.
Les potentialités d'une nouvelle entité ne peuvent pas surpasser celles du créateur.

c) Règles découlant de la création "dynamique" :

La création d'une nouvelle entité s'accompagne de la mise à disposition du créateur d'actions de délégation à quiconque avec l'entité créée comme entité-paramètre et d'actions de gratification à l'entité créée de toutes les actions statiques possédées par le créateur et enfin d'une action statique "delete" du créateur vers l'entité qu'il a créé (le créateur est en fait le possesseur).

d) Les règles de cohérence du graphe statique.

Aucune entité ne peut détenir une action sur elle-même.

e) Les règles de cohérence du graphe dynamique.

L'exécution d'une action dynamique "delegate" ou "abrogate" ne peut se faire qu'avec des entités distinctes (entité agissante, entité subissante, entité paramètre) dont l'entité agissante est possesseur de l'entité paramètre.

L'exécution d'une action dynamique "grant" ou "revoke" ne peut également se faire qu'avec trois entités distinctes.

L'exécution d'une action dynamique n'est vraiment effective que dans la mesure où il n'existe pas d'action dynamique ou statique de niveau hiérarchique plus élevé que celui de l'action dynamique ou statique qui résulterait de cette exécution.

Toutes les règles de cohérence dynamique énoncées au point 3.1.3.2. découlent des règles ci-dessus.

5.2. ANALYSE FONCTIONNELLE

Le paragraphe 5.2. reprend les résultats de l'analyse fonctionnelle et décrit une solution conceptuelle à l'aide des modèles de décomposition des traitements, de la statique et dynamique des traitements et du modèle de structuration des informations.
Ces modèles sont ceux utilisés dans le cours d'anlayse fonctionnelle donné à l'Institut d'Informatique des FNDP Namur. Avant l'exposition des résultats obtenus à l'aide de chacun des modèles, nous expliquerons brièvement chacun de ces modèles; pour de plus amples informations, nous conseillons le lecteur de se référer au livre de Messieurs Bodart et Pigneur (13).

5.2.1. LA DECOMPOSITION DES TRAITEMENTS

Le modèle de décomposition des traitements a pour objectif de fournir des critères de décomposition des traitements en traitements plus élémentaires: ces critères s'appuient sur une nomenclature hiérarchisée (projet, application, phase, fonction) dont chaque niveau assume un rôle particulier dans la conception d'un système d'information. Le "projet" est la partie d'un système d'information qui fait l'objet d'une analyse; l'"application" représente la dimension minimale d'un projet informatique; la "phase" est le niveau central et correspond à un traitement s'exécutant en un même lieu et de manière ininterrompue (dans le temps); la "fonction" est le niveau élémentaire et correspond à un seul objectif.

Le système d'autorisations comportera trois applications :

1. Application "demande-accès : modification - tables - système."

Cette application a pour but de traiter les demandes d'accès ou de modification des tables du système d'autorisations et se décompose en 4 phases :

1.1. Phase "verif. - syst."

La phase "verif-syst" teste la correction syntaxique de la demande d'accès ou de modification des tables du système.
Cette phase se décompose en cinq fonctions :

- lecture-table-entités
- lecture-table-hiérarchisation-opérations
- détermination-correction
- production-message-erreur
- production-message-correction

1.2. Phase "verif-autorisation"

La phase "verif-autorisation" teste si l'accès demandé ou modification demandée est autorisé.
Cette phase se décompose en sept fonctions :

- lecture-table-états-entités
- lecture-table-listes-utilisations
- lecture-graphe-dynamique
- lecture-graphe-statique
- détermination-autorisation
- production-message-non-autorisé
- production-message-autorisé

1.3. Phase "verif-coherence"

La phase "verif-coherence" teste si la modification demandée est telle que son exécution laisse le système dans un état respectant les règles de cohérence.

Cette phase se décompose en cinq fonctions :

- lecture-table-classes-sécurité
- lecture-table-cohérence-physique
- détermination-cohérence
- production-message-non-cohérent
- production-message-cohérent

1.4. Phase "exécution"

La phase "exécution" exécute les modifications demandées.
Cette phase se décompose en sept fonctions :

- mise-à-jour-table-entités
- mise-à-jour-table-états-entités
- mise-à-jour-table-classes-sécurité
- mise-à-jour-table-listes-utilisations
- mise-à-jour-graphe-dynamique
- mise-à-jour-graphe-statique
- mise-à-jour-table-numérotation-entités

2. Application "init-système"

L'application "init-système" a pour but d'initialiser les tables et graphes lors de l'activation du système.
Elle ne comporte qu'une phase "init-syst" qui peut être décomposée en neuf fonctions :

- creation-graphe-dynamique
- creation-graphe-statique
- creation-table-entités
- creation-table-classes-sécurité
- creation-table-états-entités
- creation-table-hiérarchisation-opération
- creation-table-numérotation-entités
- creation-table-listes-utilisations
- creation-table-cohérence.

3. Application "query-analyse"

L'application "query-analyse" a pour but d'effectuer l'analyse détaillée (cfr point 4.1.1.) des états d'autorisation et protection du système. Cette application se décompose en trois phases :

3.1. Phase "verif-syst-query"

La phase "verif-syst-query" teste la correction syntaxique de la demande au query. Elle se décompose en quatre fonctions :

- lecture-table-entités-query
- determination-correction-syntaxique
- production-message-erreur
- production-message-correction

3.2. Phase "init-query"

La phase "init-query" initialise les tables servant au query à partir de celles du système. Elle se décompose en douze fonctions :

- lecture-table-entités
- creation-table-entités-query
- lecture-table-classes-sécurité
- creation-table-classes-sécurité-query
- lecture-table-numérotation-entités
- creation-table-numérotation-entités-query
- lecture-table-états-entités
- creation-table-états-entités-query
- lecture-graphe-dynamique
- creation-graphe-dynamique-query
- lecture-graphe-statique
- creation-graphe-statique-query

3.3. Phase "recherche-réponse"

La phase "recherche-réponse" détermine les états d'autorisation et protection de façon détaillée et produit la réponse à la demande. Elle se

décompose en cinq fonctions :

- lecture-table-états-entités-query
- lecture-graphe-dynamique-query
- lecture-graphe-statique-query
- détermination-réponse
- production-message-réponse

5.2.2. LA DYNAMIQUE DES TRAITEMENTS

Le modèle dela dynamique des traitements a pour objectif de fournir des concepts et des mécanismes permettant de représenter les conditions de déclenchement, d'exécution et d'enchaînement des traitements en vue de caractériser les éléments qui causent la production des messages-résultats et les changements d'état de la mémoire du système d'information. Dans les schémas ci-dessous, les rectangles représentent des traitements, les losanges des conditions au déclenchement des traitements. les "rectangles coupés" les messages.

1. Application "demande-accès/modific. tables-syst."

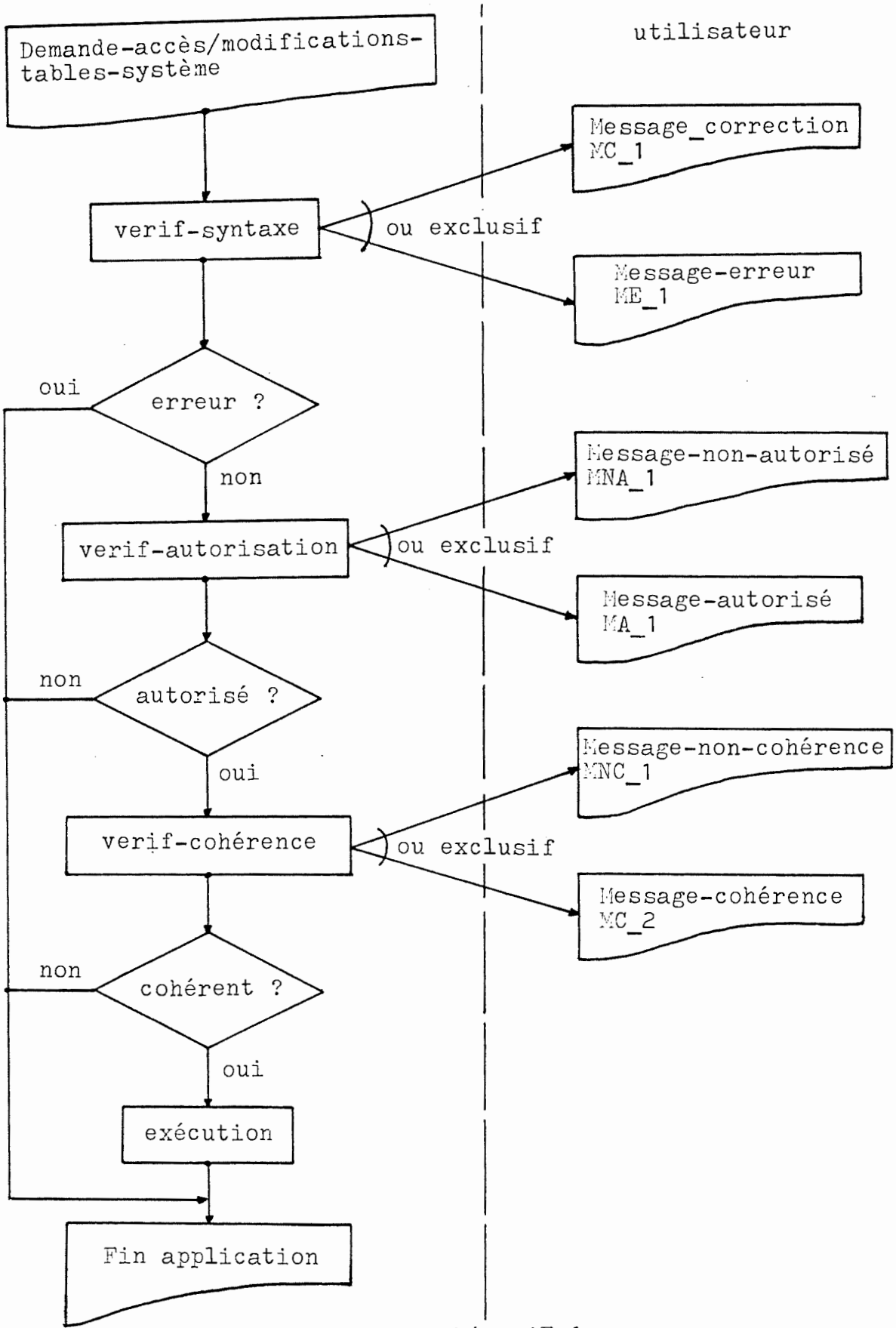


Schéma AF-1

2. Application "init-système"

Peu importe l'ordre des créations.

3. Application "query-analyse"

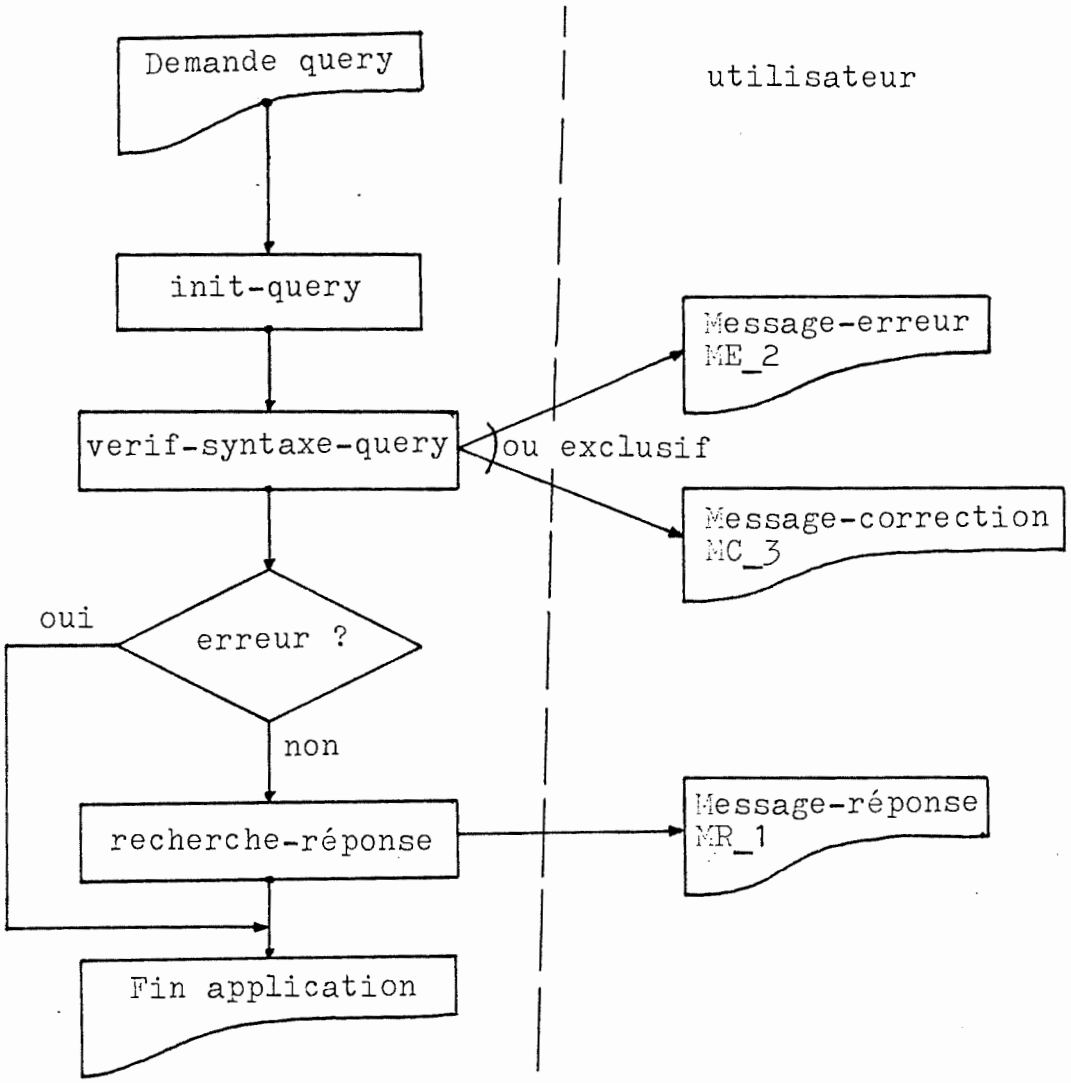


Schéma AF-2

5.2.3. LA STRUCTURATION DES INFORMATIONS

Le modèle de structuration des informations offre des concepts assistant la définition rigoureuse des informations appartenant à la mémoire du système d'information; cette définition rigoureuse permet le contrôle et un usage judicieux de ces informations.

En ce qui concerne les définitions des différents types d'entités utilisés ci-dessous, nous conseillons le lecteur de se référer à la présentation du modèle ACTEN au point 3.1. à la page 25.

a) La structuration du graphe statique

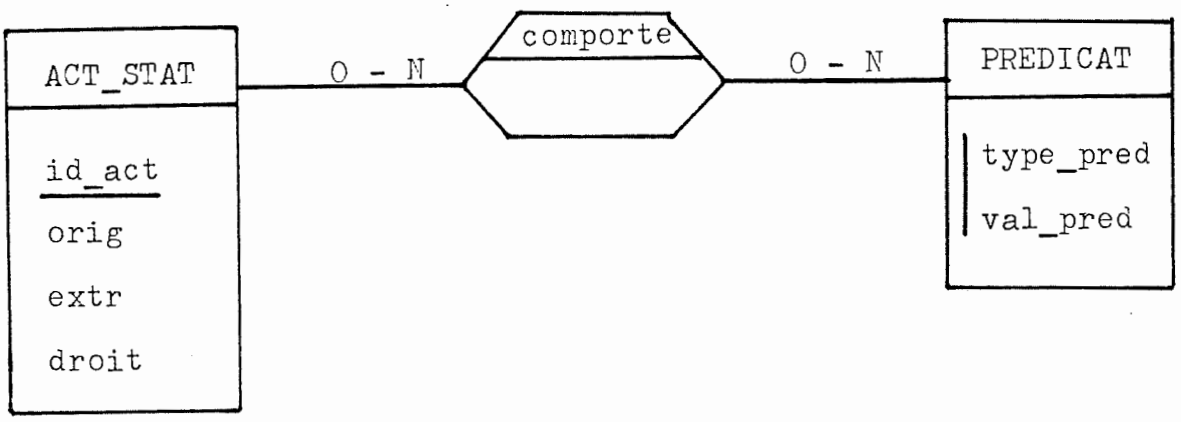


Schéma AF-3

b) La structuration du graphe dynamique

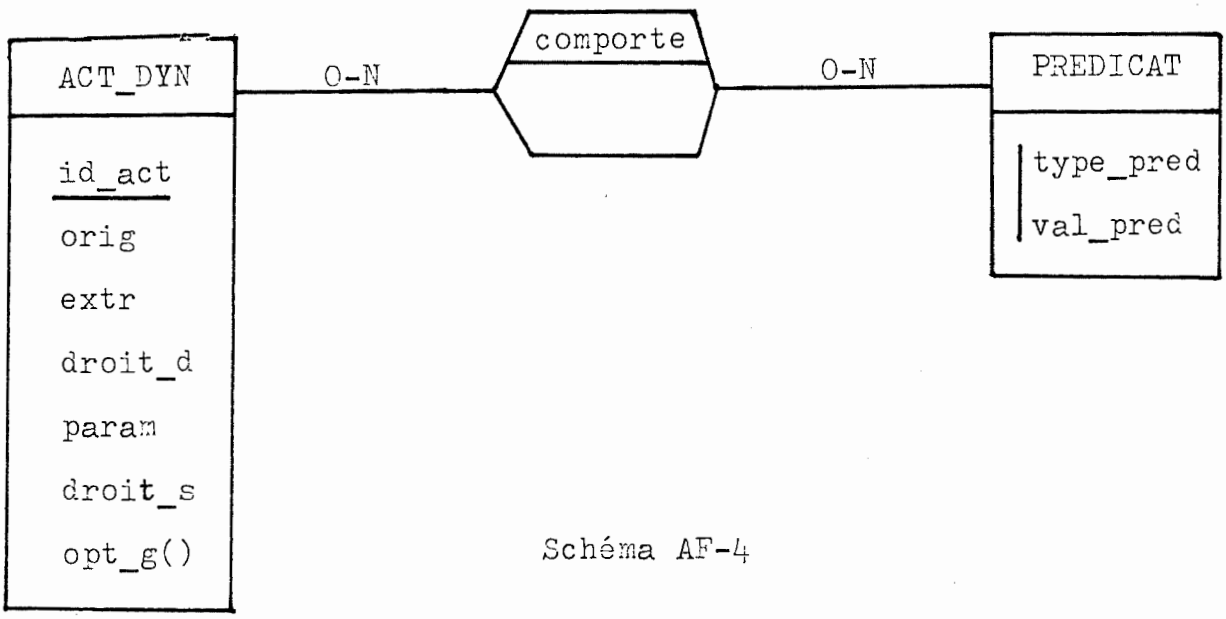


Schéma AF-4

c) La structuration de la table d'états d'entités

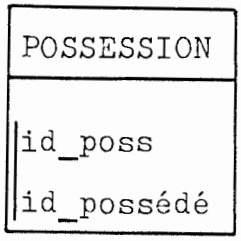


Schéma AF-5

d) La structuration de la table de classes de sécurité

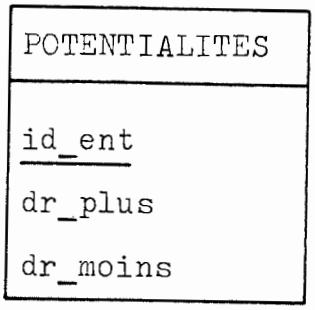


Schéma AF-6

e) La structuration de la table de hiérarchisation des opérations

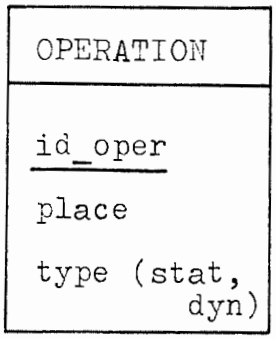


Schéma AF-7

f) La structuration de la table d'entités

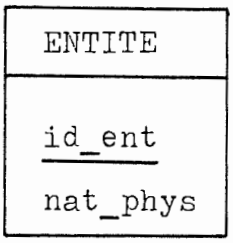


Schéma AF-8

g) La structuration de la table des listes d'utilisations

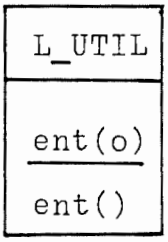


Schéma AF-9

h) La structuration de la table de cohérence physique

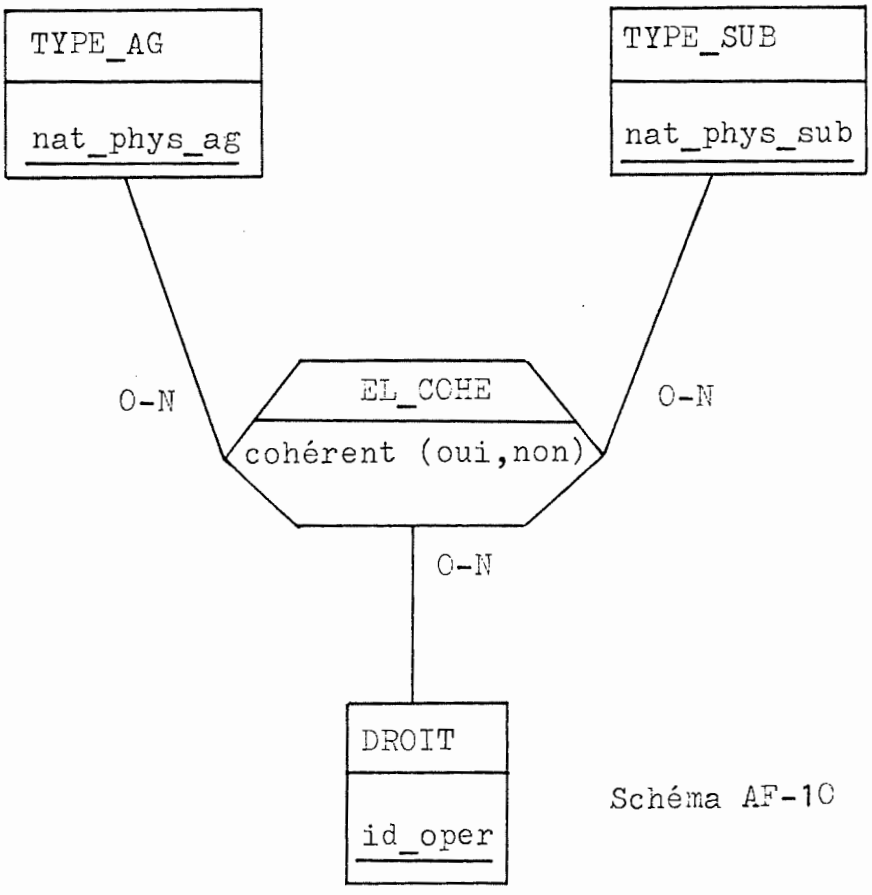


Schéma AF-10

5.2.4. LES MESSAGES

Le modèle des messages a pour objectif d'aider à la définition des messages, c'est-à-dire de leurs contenus et des supports communicationnels qu'ils utilisent.

Le modèle des messages apporte des précisions concernant les messages échangés en spécifiant les contenus et les supports de transmission.

a) Message ME - 1

Support : terminal
Contenu : - entité erronée ou droit erroné

b) Message MC - 1

Support : terminal
Contenu : "syntaxe correcte"

c) Message MA - 1

Support : terminal
Contenu : "accès autorisé"

d) Message MNA - 1

Support : terminal
Contenu : "accès non autorisé"

e) Message MNC - 1

Support : terminal
Contenu : "modifications qui laisseraient le système dans un état incohérent"

f) Message MC - 2

Support : terminal

Contenu : "règles de cohérence respectées"

g) Message ME - 2

Support : terminal
Contenu : "entité erronée"

h) Message MC - 3

Support : terminal
Contenu : "syntaxe correcte"

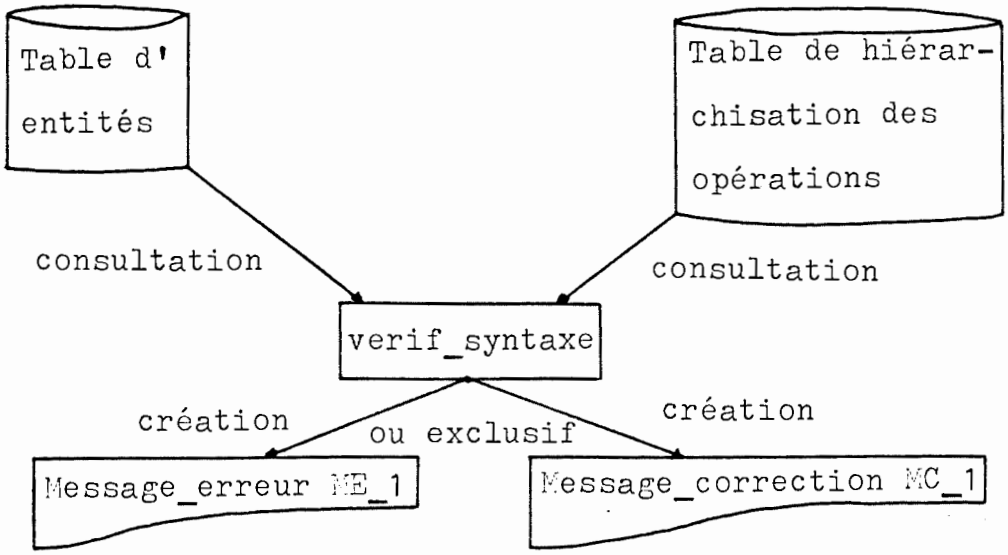
i) Message MR - 1

Support : terminal
Contenu : réponse à la demande
- liste d'entités agissantes ou subissantes
- pour chaque entité, chemin d'action et prédicats

5.2.5. LA STATIQUE DES TRAITEMENTS

Le modèle de la statique des traitements a pour objectif d'affiner l'idée de solution du problème traité en spécifiant chacun des traitements sous forme de boîte noire, c'est-à-dire en donnant les éléments de la mémoire du système d'information et les messages-données nécessaires à l'obtention des messages-résultats, en donnant les règles de traitement qui assurent l'obtention des messages-résultats, en donnant les actions élémentaires sur la mémoire du système d'information. Dans ce qui suit, un traitement est représenté par un rectangle (nous avons donné. dans ce qui suit, la statique des traitements de niveau "phase"), un message par un "rectangle coupé" et une partie de la mémoire du système par un cylindre.

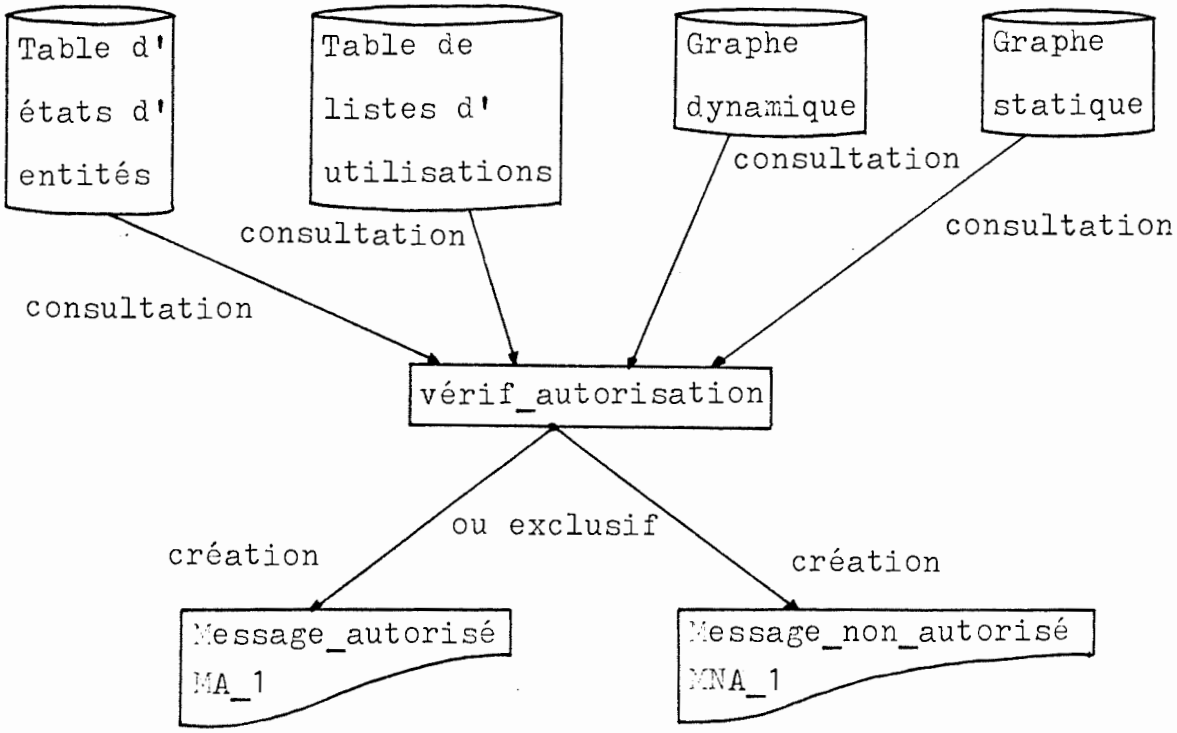
1.1. Phase "verif-synt"



Objectif : vérification syntaxique

Schéma AF-11

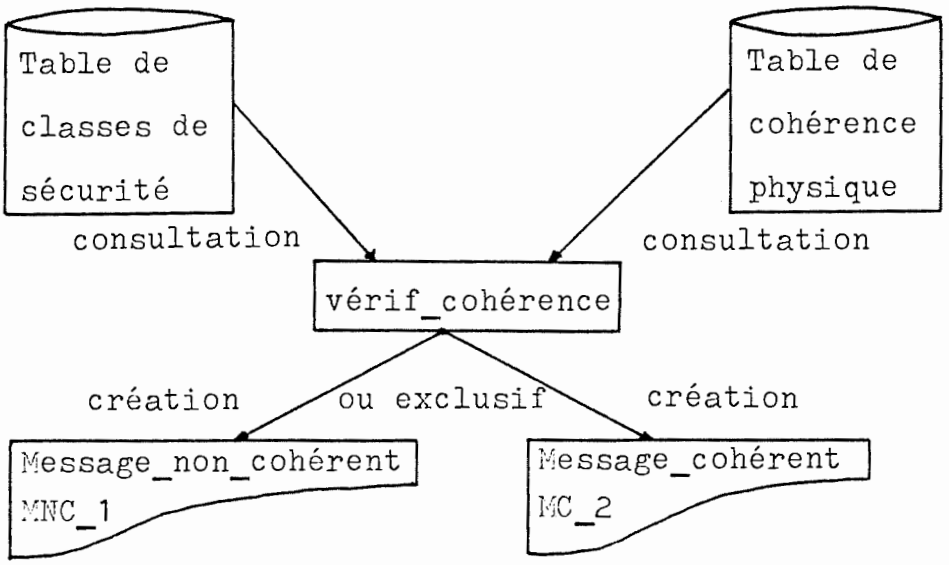
1.2. Phase "verif-autorisation"



Objectif : vérification de l' "autorisabilité" d'une action

Schéma AF-12

1.3. Phase "verif-coherence"



Objectif : vérification de la cohérence de l'insertion éventuelle d'une action dans un graphe

Schéma AF-13

1.4. Phase "exécution"

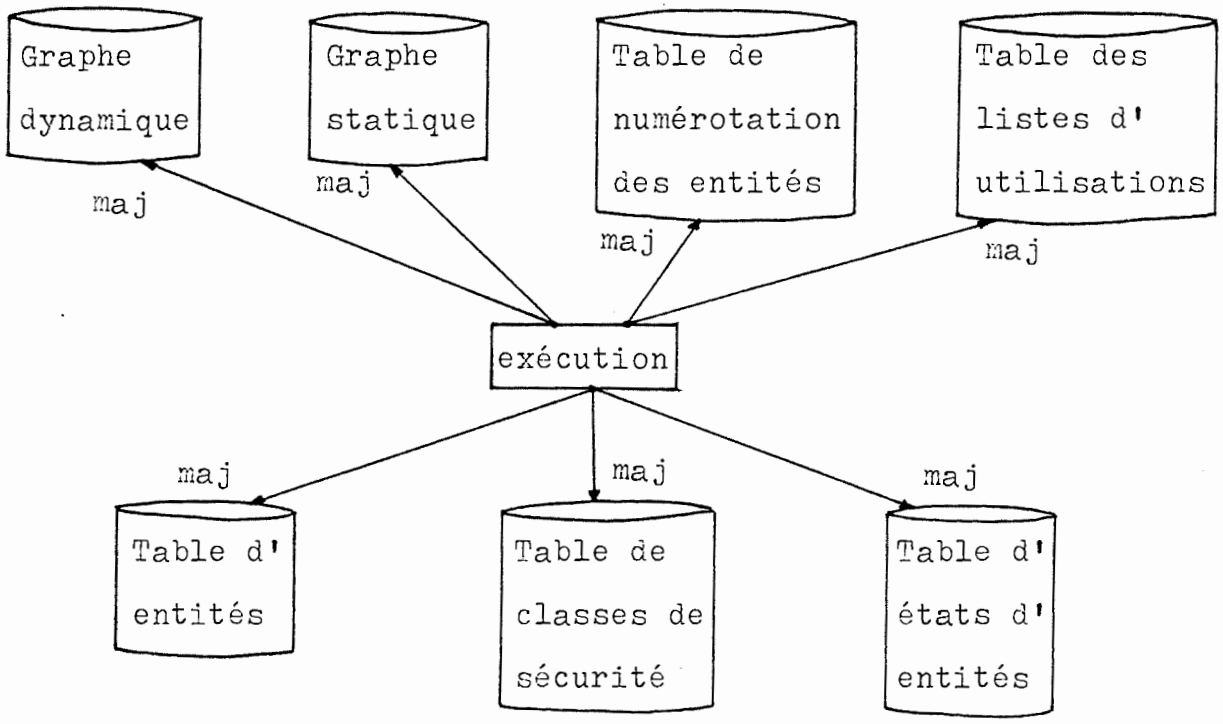
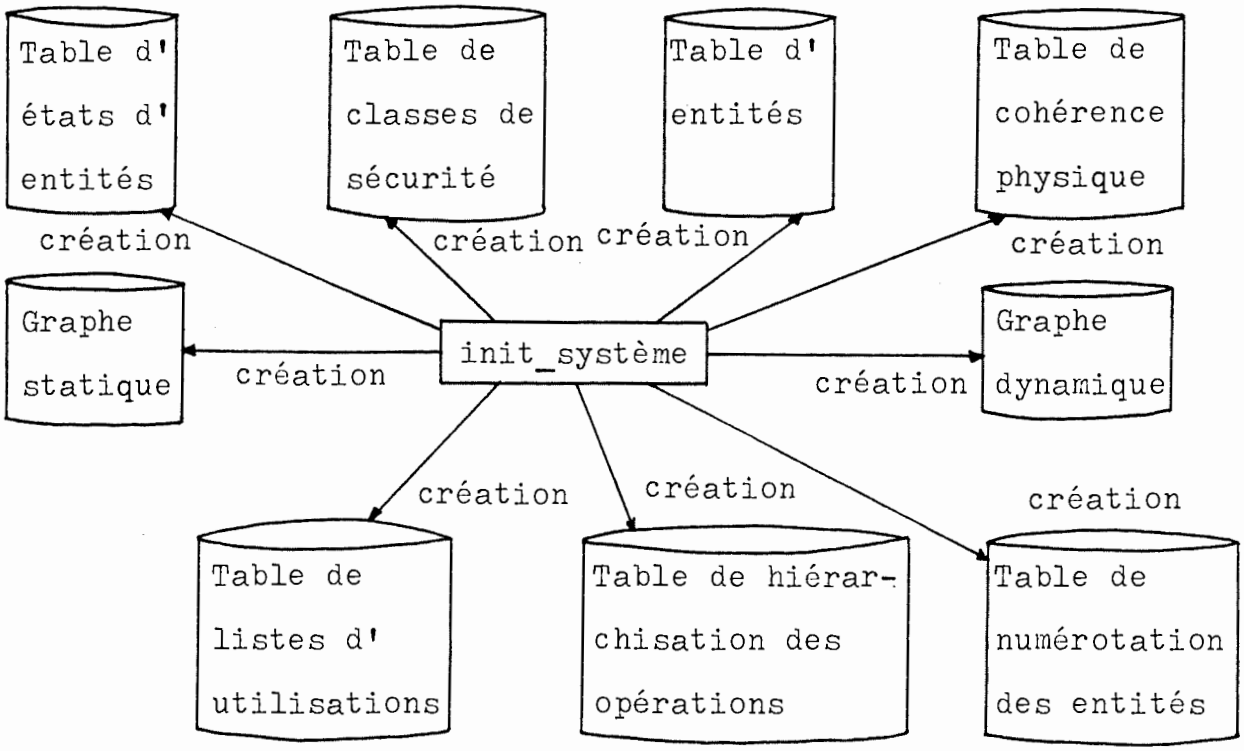


Schéma AF-14

n.b. : maj = mise à jour

Objectif : exécuter l'accès ou la modification du système demandée

2. Application "init-système"

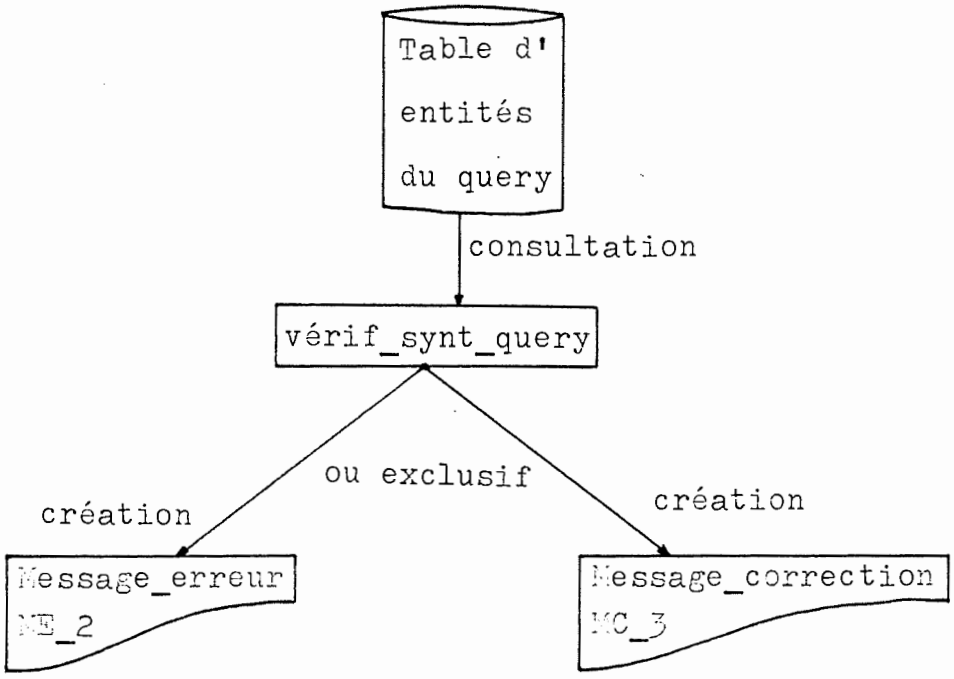


Objectif : initialiser les tables du système avec comme seule entité le "superuser"

Schéma AF-15

3. Application "query-analyse"

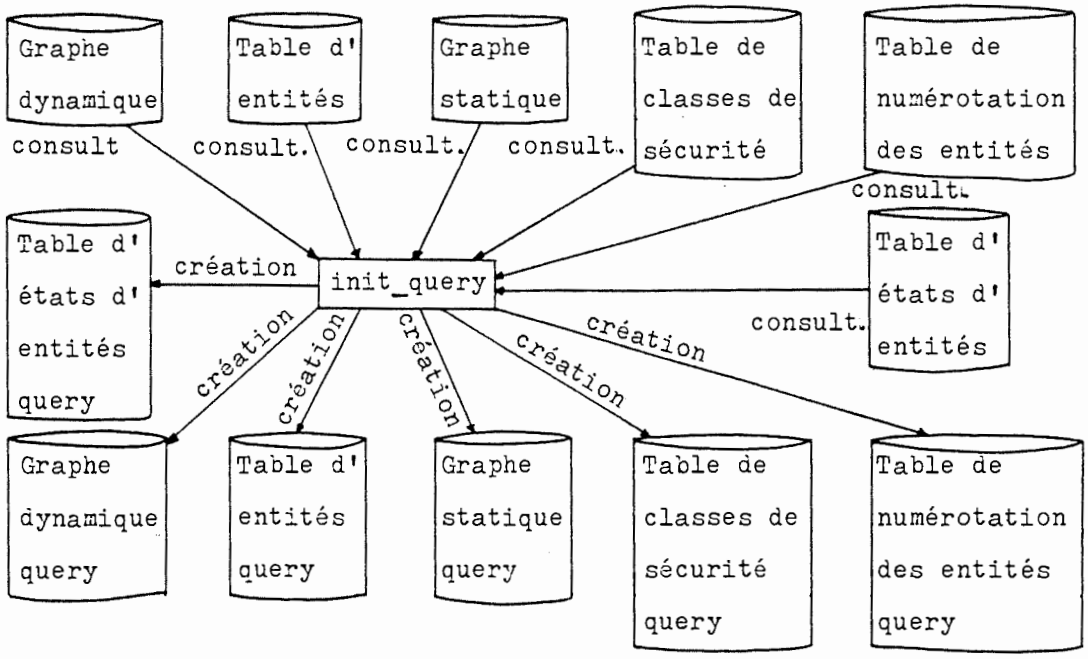
3.1. Phase "verif-synt-query"



Objectif : vérification syntaxique d'une demande au système query

Schéma AF-16

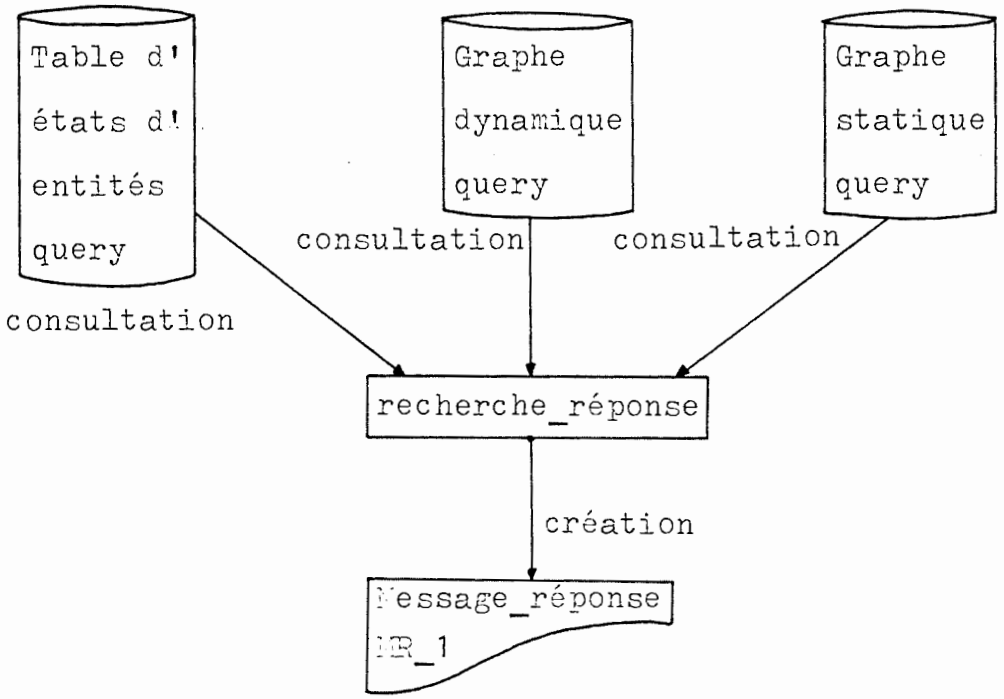
3.2. Phase "init-query"



Objectif : initialiser les tables du query à partir de celles du système réel

Schéma AF-17

3.3. Phase "recherche-réponse"

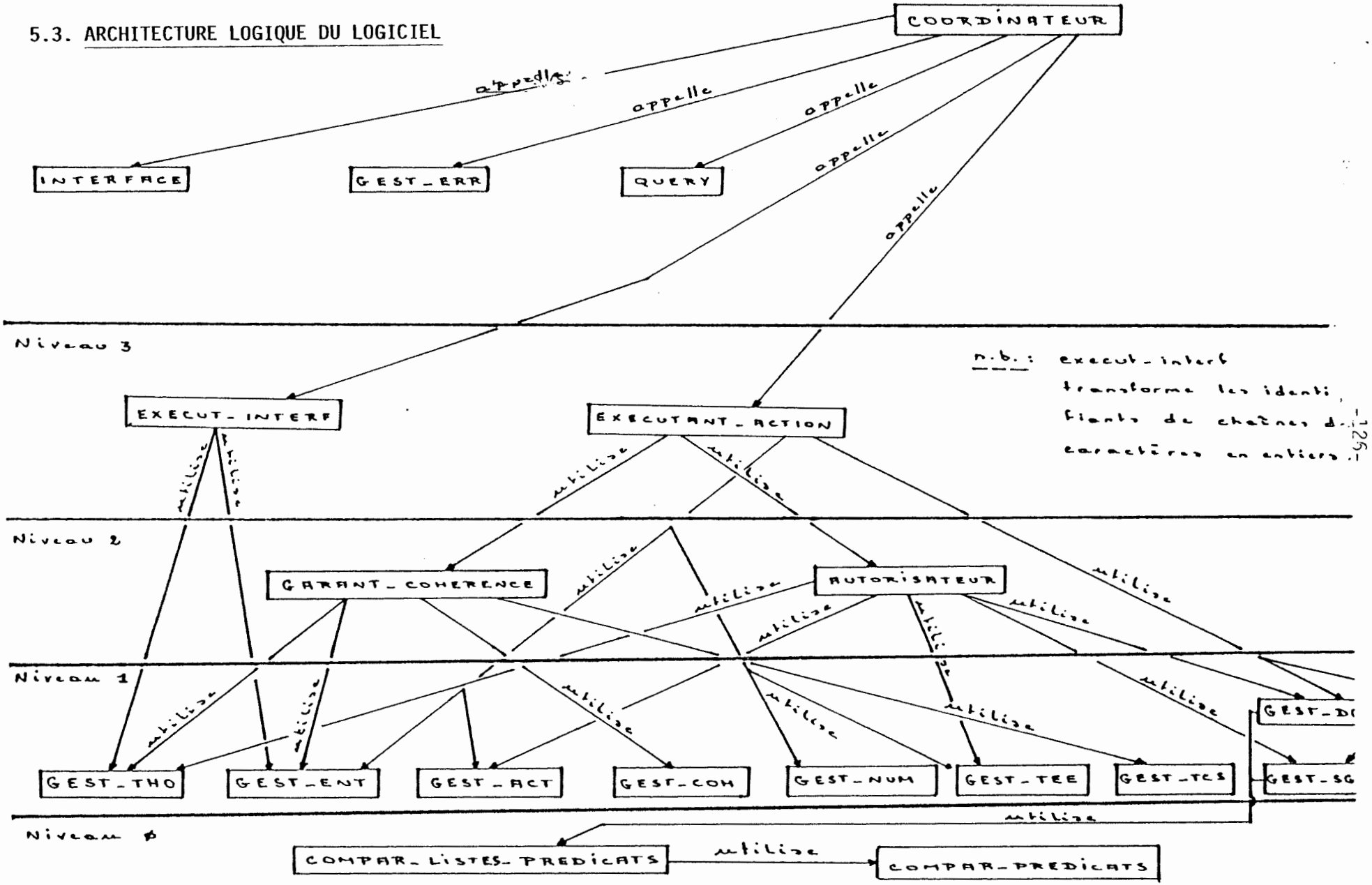


Objectif : déterminer la réponse à une demande faite au query

Schéma AF-18

Schéma L-1.

5.3. ARCHITECTURE LOGIQUE DU LOGICIEL



n.b. : execut_interf transforme les identifiants de chaînes de caractères en entiers

La relation "utilise" dont fait mention le schéma de l'architecture logique du logiciel est définie comme suit : on dit qu'un composant A d'une structure utilise un composant B de cette structure si et seulement si le fonctionnement correct de A dépend de la disponibilité d'une version correcte de B. Cette notion est issue du cours de Méthodologie de développement de logiciels donné à l'Institut d'Informatique des FNDP Namur par Monsieur Van Lamsweerde. (14).

C H A P I T R E V I

MESURES DE PERFORMANCES SUR LE LOGICIEL CONSTRUIT

Le chapitre 6 a pour objectif de traiter des mesures de performances effectuées sur le logiciel constuit: le paragraphe 6.1. expose tout d'abord les outils utilisés pour effectuer ces mesures, le paragraphe 6.2. nous donne les résultats des mesures et le paragraphe 6.3. interprète les résultats obtenus.

6.1. LES OUTILS UTILISES POUR EFFECTUER LES MESURES DE PERFORMANCES.

Les outils utilisés pour effectuer les mesures de performances sont des outils offerts par le système d'exploitation VAX/VMS. Ces outils permettent d'initialiser un timer (lib$ init-timer) qui a pour mission de retenir, entre autres, le temps total écoulé depuis l'initialisation, et de visualiser le contenu du timer (lib $ show-timer).

6.2. LES RESULTATS DES MESURES.

Nous avons effectué les mesures du temps total écoulé, soit du temps de réponse, entre une demande de l'utilisateur et le moment où la réponse lui parvient. Ces mesures ont été effectuées à faible charge, avec un graphe statique rempli d'une trentaine de droits et une chaîne d'utilisation de 4 entités. Dans de telles conditions, nous avons relevé que le temps total écoulé entre une demande d'accès et l'arrivée de la réponse est en moyenne de 10 centièmes de seconde.

6.3. INTERPRETATION DES RESULTATS.

Les mesures effectuées ont pour contexte un graphe statique rempli d'une trentaine de droits, ce qui est peu pour un système d'autorisations en plein exercice de ses fonctions; si l'on prend comme référence un graphe statique comportant 200 droits (50 entités ayant chacune 4 droits d'action), on peut évaluer la moyenne du temps de réponse à l'aide de la notion de complexité théorique d'un algorithme. Cette notion est issue du cours de Théorie des graphes donné par Monsieur Fichefet à l'Institut d'Informatique

des FNDP Namur (15) et exprime la façon dont varie le temps de calcul d'un algorithme en fonction d'un paramètre. Dans notre cas, nous affirmons que le temps de calcul de notre logiciel en fonction de la taille n du graphe statique est de l'ordre de 2n (pour une moyenne de 2 prédicats) il s'ensuit que si nous multiplions le temps de réponse par (200/30) multiplié par 2, le temps de réponse moyen de notre logiciel monte alors à 1,3 seconde.

Etant donné cette moyenne de temps de réponse de 1,3 seconde, le fait que l'on a travaillé à faible charge, et le fait que l'on a omis les problèmes de concurrence, nous affirmons que le plus apporté (par rapport aux systèmes de sécurité prévalant sur UNIX ou sur VAX/VMS) par l'introduction de prédicats dans un système tel que celui-ci risque, en utilisation réelle, de grever fortement les temps de réponses du système.

C O N C L U S I O N

Le travail effectué a permis de se rendre compte que les problèmes de sécurité dans un système informatique sont bien réels et loin d'être aisés à résoudre (même pour le domaine des systèmes d'autorisations que l'on pourrait à priori croire plus malléable) et forcent en fait une solution en forme de compromis (entre transparence et efficacité) plutôt qu'une solution réelle et unique. Ce compromis est de plus influencé par les autres éléments de sécurité que l'on veut mettre en oeuvre ainsi que du contexte d'utilisation du système informatique (à des fin militaires, médicales, commerciales ...).

Bien que n'ayant ici parlé que de système d'autorisations, nous insistons sur le fait que ces derniers ne sont qu'une mesure de sécurité parmi d'autres et doivent être intégrés dans une politique de sécurité générale.

ANNEXES

ANNEXE A

Soit une chaîne d'opérations comportant n entités intermédiaires telle que représentée par la figure A-1.

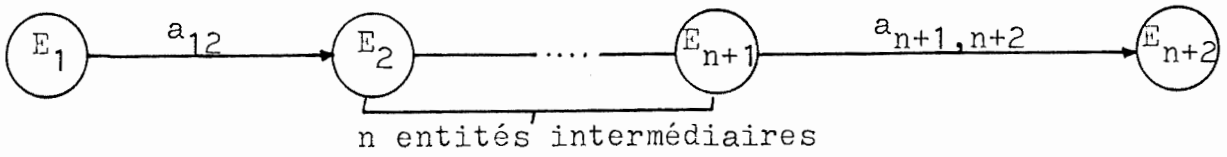


Figure A-1. Chaîne d'opérations comportant n entités intermédiaires

Théorème

L'opération indirecte maximale que E_1 peut avoir sur E_{n+2} est l'opération obtenue en combinant par l'algorithme A l'opération $a_{1,n+I}$ et l'opération $a_{n+1,n+2}$.

N.B.. les prédicats résultant étant les mêmes, on peut ici confondre action et opération.

Démonstration (par induction)

I-. Vrai pour k = 2 ?

La figure A-2. représente les deux façons possibles de combiner les opérations pour obtenir l'opération indirecte a_{14}.

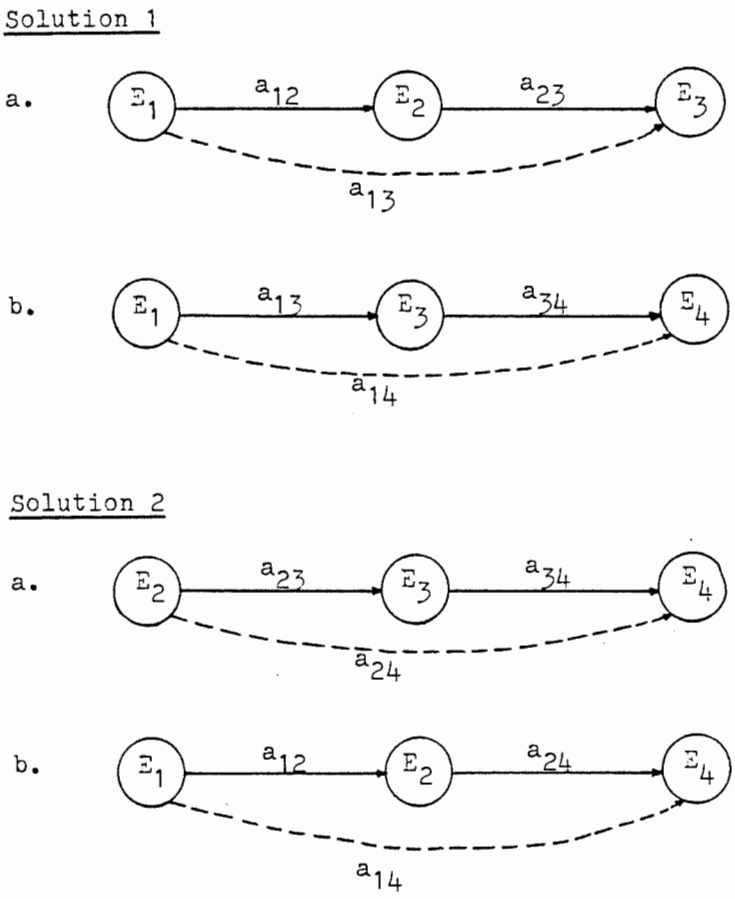


Figure A-2. Deux façons possibles de combiner les opérations pour obtenir l'opération indirecte a_{14}

Thèse : $L(a_{14}$, solution I) $\gtrsim L(a_{14}$, solution 2).

Au vu de l'algorithme A, on peut affirmer que dans le cas de la solution I, le point a- est inutile car ne sert pas au point b- (l'application de l'algorithme A au point b- ne tient compte que de a_I^+ et a_{34}).
Six cas sont à considérer :

1. $L(a_I^+) \gtrsim L(a_{34}) \gtrsim L(a_2^+)$

   Solution 1 : b- $L(a_{14}) = L(a_{34})$

   Solution 2 : a- $L(a_{24}) = L(a_2^+)$
   b- $L(a_{14}) = L(a_2^+)$

   Or, $L(a_{34}) \gtrsim L(a_2^+) \longrightarrow$ OK.

2. $L(a_1^+) \geq L(a_2^+) \geq L(a_{34})$.

Solution 1 : b- $L(a_{14}) = L(a_{34})$

Solution 2 : a- $L(a_{24}) = L(a_{34})$

b- $L(a_{14}) = L(a_{34})$

$\longrightarrow$ OK.

3. $L(a_2^+) \geq L(a_1^+) \geq L(a_{34})$

Solution 1 : b- $L(a_{14}) = L(a_{34})$

Solution 2 : a- $L(a_{24}) = L(a_{34})$

b- $L(a_{14}) = L(a_{34})$

$\longrightarrow$ OK.

4. $L(a_2^+) \geq L(a_{34}) \geq L(a_1^+)$

Solution 1 : b- $L(a_{14}) = L(a_1^+)$

Solution 2 : a- $L(a_{24}) = L(a_{34})$

b- $L(a_{14}) = L(a_1^+)$

$\longrightarrow$ OK.

5. $L(a_{34}) \geq L(a_1^+) \geq L(a_2^+)$

Solution 1 : b- $L(a_{14}) = L(a_1^+)$

Solution 2 : a- $L(a_{24}) = L(a_2^+)$

b- $L(a_{14}) = L(a_2^+)$

Or, $L(a_1^+) \geq L(a_2^+)$ ⟶ OK.

6. $L(a_{34}) \geq L(a_2^+) \geq L(a_1^+)$

Solution 1 : b- $L(a_{14}) = L(a_1^+)$

Solution 2 : a- $L(a_{24}) = L(a_2^+)$

b- $L(a_{14}) = L(a_1^+)$

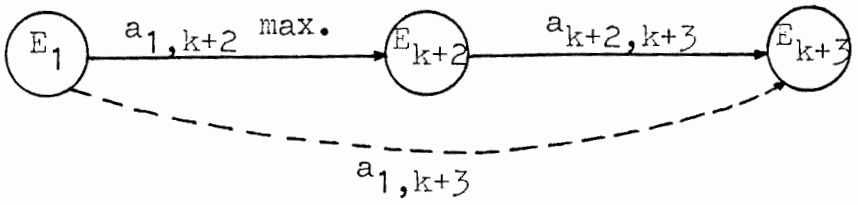
⟶ OK.

Donc, vrai pour k = 2.

II-. <u>Vrai pour n = k ⟹ vrai pour n = k+1 ?</u>

La figure A-3. représente les deux façons de combiner les opérations pour obtenir l'opération indirecte a_{14}, en supposant que la façon dont on a combiné jusqu'ici pour n = k, K-1, ...,2 est celle donnant les opérations indirectes maximales.

Solution 1



Solution 2

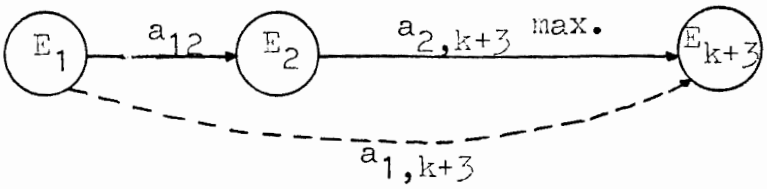


Figure A-3. Deux façons de combiner les opérations pour obtenir l'opération indirecte $a_{1,k+3}$

Se référant à la solution 2 de la figure A-3., on peut affirmer que $L(a_{k+2,k+3}) \geq L(a_{2,k+3})$ car pour déterminer $a_{2,k+3}$ de façon maximale on a dû prendre le minimum de $L(a_2^+)$ et $L(a_{k+2,k+3})$.
Puisque $L(a_{k+2,k+3}) \geq L(a_{2,k+3})$, il s'ensuit que les deux applications de l'algorithme A donneront $L(a_{1,k+3}$; solution 1) $\geq L(a_{1,k+3}$; solution 2), ce qui démontre le théorème.

ANNEXE B - LISTE DES ABREVIATIONS

1. Sa (Ei) = état d'autorisation de l'entité Ei.
2. Sp (Ei) = état de protection de l'entité Ei.
3. SGDA = sous-graphe direct d'analyse d'état d'autorisation.
4. SGDP = sous-graphe direct d'analyse d'état de protection.
5. n.h.i. = "est de niveau hiérarchique inférieur à".
6. n.h.é. = "est de niveau hiérarchique égal à".
7. n.h.s. = "est de niveau hiérarchique supérieur à".
8. n.h.inc. = "est de niveau hiérarchique incomparable à"
9. SO = "opération statique"
10. DO = "opération dynamique"
11. SA = "action statique"
12. DA = "action dynamique"

ANNEXE C - BIBLIOGRAPHIE

1. Principles of Data Security (chapitre 3)
   Ernst L. Leiss
   Foundations of computer science
   Series Editor : Raymond E. Miller
   Georgia Institute of Technology
   Plenum Publishing Corporation

2. Acten : A conceptual Model for Security Systems design
   M. Fugini, G. Martella
   Computers and Security
   Official journal of IFID TC-11
   volume 3, number 3, August 1984

3. Protection in Operating Systems
   M.A. Harrison, W.L. Ruzzo, J.D. Ullman
   Communications ACM 19 (8), pp. 461 - 471 (August 1976)

4. A linear time algorithm for deciding subject security
   ACM 24, 3, July 1977, pp. 453 - 464

5. Unidirectional transport of rights and Tome - Grant control
   A. Lockman, N. Ninsmy
   IEEE Transactions on Software Engineering
   Vol. 8. n°6. november 1987

6. Granting and revoking discretionary authority
   Larson J.R.
   Information Systems. vol. 8, n°4, 1983

7. Treating data privacy in distributed systems
   Bussolati U., Martella G.
   Information and Management, Vol. 4, n°6, 1981

8. Data security management in distributed data bases
Bussolati U., Martella G.
Information Systems, Vol. 7, n°3, 1982

9. Sécurité des systèmes informatiques. notes de cours
Ramaekers J.
Mai 1984

10. Guide to VAX/VMS version 4.2. System Security digital equipment corporation

11. Unix system security
Wood P.H. Kochan S.G.
Hayder Book Compagny

12. Le language C
Kernighan B.W. Ritchie D.M. (Traduction : Buffenoir T.)
Masson 1984

13. Conception assistée des applications informatiques
1. Etude d'opportunité et analyse conceptuelle
F. Bodart, Y. Pigneur
Masson (Méthode + Programmes)

14. Méthodologie de développement de logiciels. notes de cours
A. Van Lamsweerde

15. Théorie des graphes et son algorithmique, notes de cours
J. Fichefet

P A R T I E P R A T I Q U E

DEVELOPPEMENT D'UN SYSTEME D'AUTORISATIONS ISSU DU MODELE A C T E N

TABLE DES MATIERES

C H A P I T R E I

DEVELOPPEMENT DU LOGICIEL

Le présent chapitre fait état de commentaires concernant la construction des divers modules composant ce logiciel. Il permet en fait au lecteur de suivre à la trace le processus de conception de ces modules. L'ordonnancement des commentaires est tel que l'on part des niveaux hiérarchiques les plus bas vers les niveaux hiérarchiques les plus élevés.

1. Démarche de conception des modules de niveau 0.

(compar)

COMMENTAIRES SUR COMPAR.C

Le niveau 0 de la hierarchisation de ce logiciel a pour mission d'implementer la notion de relation de contraignance entre deux ensembles de predicats.

Pour rappel(cfr 1er livre),cette notion,nee du concept de hierarchisation des actions,vise a comparer les pouvoirs restrictifs de deux ensembles de predicats sur l'execution d'une operation.

Les valeurs_resultats d'une comparaison de deux ensembles de predicats Ep1 et Ep2 peuvent etre les suivantes:

'<' : Ep1 est moins contraignant que Ep2
'>' : Ep1 est plus contraignant que Ep2
'=' : Ep1 est aussi contraignant que Ep2
'<>' : Ep1 exerce une contrainte incomparable a celle exercee par Ep2

1 _ Le module comp_lp

1_1.Specifications.

Arg. : Ep1,Ep2 : deux ensembles de predicats

Pre. : -----

Result.: res : code_resultat

Post. : res vaut

'<' si Ep1 est moins contraignant que Ep2
'>' si Ep1 est plus contraignant que Ep2
'=' si Ep1 est aussi contraignant que Ep2
'<>' si Ep1 exerce une contrainte incomparable a celle exercee par Ep2.

1_2.Indications sur la conception de l'algorithme.

Soit donc a construire un algorithme A repondant aux specifications enoncees au point 1_1.

a_ Raisonnement par decomposition et analyse de precedence.

La notion de relation de contraignance entre deux ensembles de predicats fait intervenir une relation de contraignance entre toute paire de predicats (P1,P2) telle que P1 appartient a Ep1 et P2 appartient a Ep2; elle induit donc une decomposition en trois sous_algorithmes :

1_ Sous_algorithme AA qui implemente la comparaison de toutes les paires (P1,P2) telle que P1 appartient a Ep1 et P2 appartient a Ep2.

Ce sous_algorithme sera donc charge de remplir une structure intermediaire st_int1 avec les resultats de ces differentes comparaisons :

st_int1[i,j] = resultat de la comparaison du ieme predicat de Ep1 avec le jeme predicat de Ep2.

2_ Sous_algorithme AB qui implemente une fonction d'epuration des resultats obtenus en 1_ (cfr 1er livre : notion de relation de contraignance).

Ce sous_algorithme determine donc pour chaque predicat P1 de Ep1 quelles sont les relations que P1 entretient avec un predicat quelconque de Ep2.
Par elimination des relations non presentes pour tous les predicats P1 de Ep1 avec Ep2,ce sous_algorithme determine finalement quelles sont les relations de contraignance possibles entre Ep1 et Ep2.
Il produit donc une seconde structure intermediaire st_int2 donnant toutes les relations de contraignance possibles entre Ep1 et Ep2 :

st_int2[i] = ieme relation de contraignance possible.

3_ Sous_algorithme AC qui, partant des resultats obtenus en 2_,determine finalement la relation de contraignance entre Ep1 et Ep2 et met ces resultats dans 'result'.

b_ Construction de AA.
^^^^^^^^^^^^^^^^^^^^^^

Raisonnement par induction --> double structure iterative (d'abord sur les elements de Ep1, puis de facon imbriquee sur les elements de Ep2).

Definition d'un sous_module comp_p de comparaison de deux predicats (cfr specifications au point 2_1.)

c_ Construction de AB.
^^^^^^^^^^^^^^^^^^^^^^

On a deja sous_entendu un raisonnement par decomposition pour construire AB ci_dessus au point 1_3.a.
Les deux taches a executer sequentiellement sont :

1_ Determination pour chaque predicat P1 de Ep1 des relations de contraignance qu'il entretient avec un predicat quelconque de Ep2.
(sous_algorithme ABA)

2_ Determination de toutes les relations de contraignance possibles entre Ep1 et Ep2 (sous_algorithme ABB).

d_ Construction de AC.
^^^^^^^^^^^^^^^^^^^^^^

Analyse par cas en fonction :

1_ du nombre de relations distinctes possibles entre Ep1 et Ep2.

2_ des relations possibles elles_memes.

e_ Construction de ABA.
^^^^^^^^^^^^^^^^^^^^^^

Soit donc un predicat P1 de Ep1;il s'agit,partant des resultats de comparaison de P1 avec chacun des predicats de Ep2,de determiner les relations que P1 entretient avec un predicat quelconque de Ep2.

Il suffit donc de retenir chaque valeur distincte apparue dans l'ensemble des comparaisons (P1,elem[Ep2]) et differente de '<>' (car '<>' signifie qu'on n'arrive pas a determiner de relation),et ce,pour tout P1 de Ep1.

ex.: si '<' est le resultat de comp_p(elem i de Ep1,elem j de Ep2) et de comp_p(elem i de Ep1,elem k de Ep2),on ne retiendra qu'une seule fois '<' pour le ieme predicat de Ep1 --> structure st_int11.

f_ Construction de ABB.
^^^^^^^^^^^^^^^^^^^^^^^

Raisonnement par induction sur le nombre de predicats de Ep1 et sur l'ensemble des resultats de comparaison possibles ('<','>','=','<>') --> struct.iterative

pour tout resultat possible,si ce resultat apparait comme valeur distincte (cfr ABA) pour tout P1 de Ep1,on retient ce resultat,sinon on l'ecarte

--> st_int2.

1_3.Algorithme.

Debut

```
pour tout predicat Ep1[i] de Ep1                                  <--|
   |                                                                |
   | pour tout predicat Ep2[j] de Ep2                               | AA
   |    |                                                           |
   |    | st_int1[i,j] = comp_p(Ep1[i],Ep2[j])                      <--|

pour toute ligne i de st_int1                                     <--|
   |                                                                |
   | pour toute colonne j de st_int1[i,]                            |
   |    |                                                           | ABA
   |    | si    st_int1[i,j] n'appartient pas a st_int11[i,] et     |
   |    |       st_int1[i,j] est different de '<>'                  |
   |    | alors ajouter st_int1[i,j] dans st_int11[i,]              <--|

pour toute valeur_resultat v[k] possible ('<','>','=','<>')      <--|
   |                                                                |
   | ajouter v[k] dans st_int2                                      |
   | pour toute ligne i de st_int11  (b1)                           |
   |    |                                                           |
   |    | trouve <-- non                                            |
   |    | pour toute colonne j de st_int11  (b2)                    |
   |    |    |                                                      | ABB
   |    |    | si    v[k] = st_int11[i,j]                           |
   |    |    | alors |                                              |
   |    |    |       | trouve <-- oui                               |
   |    |    |       | sortir (b2)                                  |
   |    |    |       |                                              |
   |    | si    trouve = non                                        |
```

```
   |    |       | retirer v[k] de st_int2                                  |
   |    |       | sortir (b1)                                        <--|

si    ||st_int2|| = 0                                                 <--|
alors |                                                                   |
      | si    (||Ep1|| = 0 et ||Ep2|| = 0)                               |
      | alors result <-- '='                                            |
      | si    (||Ep1|| = 0 et ||Ep2|| <> 0)                              |
      | alors result <-- '<'                                            |
      | si    (||Ep1|| <> 0 et ||Ep2|| = 0)                              |
      | alors result <-- '>'                                            |
      | si    (||Ep1|| <> 0 et ||Ep2|| <> 0)                             |
      | alors result <-- '<>'                                           |
      |                                                                 |
si    ||st_int2|| = 1                                                 |
alors result <-- st_int2[1]                                           | AC
si    ||st_int2|| = 3                                                 |
alors result <-- '='                                                  |
si    ||st_int2|| = 2                                                 |
alors |                                                               |
      | si    ('<' et '>') appartiennent a st_int2                      |
      | alors result <-- '='                                          |
      | si    ('<' et '=') appartiennent a st_int2                      |
      | alors result <-- '<'                                          |
      | si    ('>' et '=') appartiennent a st_int2                      |
      | alors result <-- '>'                                          |
      |                                                           <--|
Fin.
----
```

1_4.Passage module logique --> module physique.

a_ Codage du resultat en entier.

```
'<'  -->  0
'>'  -->  1
'='  -->  2
'<>' -->  3
```

b_ Optimisation.

La partie ABA de l'algorithme est incorporee dans la partie AA : avant d'inserer un nouveau resultat de comparaison,on effectue les tests de l'algorithme ABA et,en cas de test fructueux,on omet l'insertion.

Il en resulte l'elimination de la structure intermediaire st_int1 et AA produit directement la structure intermediaire st_int11.

D'autre part,on prendra pour st_int2 la premiere ligne de st_int11 (apres traitement par ABA,st_int2 correspondra a la pramiere ligne de st_int11).

c_ Choix de representation des predicats.

1_ Limitation du nombre de predicats d'un ensemble de predicats :

2_ Representation d'un predicat :

sous forme de structure C comprenant les champs :

. code_pr : indique le type du predicat (entier)

si 0 --> predicat portant sur l'existence d'une action dynamique sans predicat.

si 1 --> predicat portant sur l'existence d'une action statique sans predicat.

si 2 --> predicat temporel.

si 3 --> predicat portant sur la nature physique d'une entite.

si 4 --> predicat portant sur la potentialite active d'une entite.

si 5 --> predicat portant sur la potentialite passive d'une entite.

si 6 --> predicat portant sur une relation de possession entre deux entites.

. si code_pr = 0 --> les champs suivants sont significatifs :

- ent1_p : identifiant de l'entite agissante
- ent2_p : identifiant de l'entite subissante
- droit_p : identifiant de l'operation dynamique
- param_p : identifiant de l'entite parametre
- droit_ps : identifiant de l'operation statique

. si code_pr = 1 --> les champs suivants sont significatifs :

- ent1_p : identifiant de l'entite agissante
- ent2_p : identifiant de l'entite subissante
- droit_p : identifiant de l'operation statique

. si code_pr = 2 --> les champs suivants sont significatifs :

- h_d : heure de debut
- h_f : heure de fin

. si code_pr = 3 --> les champs suivants sont significatifs :

- ent1_p : identifiant de l'entite
- nat_phys : nature physique minimum

. si code_pr = 4 --> les champs suivants sont significatifs :

- ent1_p : identifiant de l'entite
- droit_p : potentialite active minimum

. si code_pr = 5 --> les champs suivants sont significatifs :

- ent1_p : identifiant de l'entite
- droit_p : potentialite passive minimum

. si code_pr = 6 --> les champs suivants sont significatifs :

- ent1_p : identifiant de l'entite possesseur
- ent2_p : identifiant de l'entite possedee

3_ Ensemble de predicats --> n_p : nombre de predicats de t_p
t_p : tableau de predicats

d_ Structures intermediaires
^^^^^^^^^^^^^^^^^^^^^^^^^^^^

st_int1 --> tableau result[L_P,L_P] d'entiers +
tableau nbr_j[L_P] d'entiers (a initialiser)
avec nbr_j[i] indiquant a tout moment le nombre
d'elements contenus dans la ieme ligne de result.

st_int2 --> result[0,L_P] + nbr_j[0]

n.b.: les tableaux C commencent avec l'indice valant 0.

2_ Le module comp_p.

2_1.Specifications.

Arg. : P1,P2 : deux predicats
^^^^^^^^

Pre. : (0 <= P1.code_pr <= 6 et 0 <= P2.code_pr <= 6)
^^^^^^^^

Result.: res : code_resultat
^^^^^^^^

Post. : res vaut
^^^^^^^^

'<' si P1 est moins contraignant que P2
'>' si P1 est plus contraignant que P2
'=' si P1 est aussi contraignant que P2
'<>' si P1 exerce une contrainte incomparable a celle
exercee par P2.

2_2.Indications sur la conception de l'algorithme.

Soit donc a construire un algorithme B repondant aux specifications enoncees au point 2_1.

a_ Analyse par cas sur les types de predicats.
^^

Soit P1 et P2 sont de types distincts auquel cas res prendra la valeur '<>',soit ils sont de meme type auquel cas il faudra pousser les investigations plus loin en fonction du type des predicats.

D'ou la structure conditionnelle suivante :

P1 est de meme type que P2 <--|

```
   sinon  res <-- '<>'                                              <--|

b_ Construction de BA.
^^^^^^^^^^^^^^^^^^^^^^

Analyse par cas sur le type des predicats :

    si     type = existence_action_dynamique                        <--|
    alors  si     (memes entite agissante,entite subissante,entite     |
                   parametre)                                          |
           alors  comparer encore plus avant (BAA)                     |
           sinon  res <-- '<>'                                         |
    si     type = existence_action_statique                           |
    alors  si     (memes entite agissante,entite subissante)           |
           alors  comparer encore plus avant (BAB)                     |
           sinon  res <-- '<>'                                         |
    si     type = temporel                                            |
    alors  si     ( h_d de P1 < h_d de P2 )                           |
           alors  si     ( h_f de P1 < h_f de P2 )                    |
                  alors  res <-- '<>'                                  |
                  sinon  res <-- '<'                                   |
           sinon  si     ( h_d de P1 > h_d de P2 )                    |
                  alors  si     ( h_f de P1 > h_f de P2 )              | BA
                         alors  res <-- '<>'                           |
                         sinon  res <-- '>'                            |
                  sinon  si     ( h_f de P1 < h_f de P2 )              |
                         alors  res <-- '>'                            |
                         sinon  si     ( h_f de P1 > h_f de P2 )       |
                                alors  res <-- '<'                     |
                                sinon  res <-- '='                     |
    si     type = nature_physique                                     |
    alors  si     ( meme entite )                                     |
           alors  comparer plus loin (BAC)                            |
           sinon  res <-- '<>'                                        |
    si     type = potentialite_active ou type = potentialite_passive  |
    alors  si     ( meme entite )                                     |
           alors  comparer plus loin (BAD)                            |
           sinon  res <-- '<>'                                        |
    si     type = possession                                          |
    alors  si     ( memes entites possesseur et possedee )            |
           alors  res <-- '='                                         |
           sinon  res <-- '<>'                                      <--|

c_ Construction de BAA.
^^^^^^^^^^^^^^^^^^^^^^^

Analyse par cas en fonction des comparaisons hierarchiques des operations
dynamiques et statiques.

d_ Construction de BAB.
^^^^^^^^^^^^^^^^^^^^^^^

Analyse par cas en fonction de la comparaison hierarchique des operations
statiques.

e_ Construction de BAC.
^^^^^^^^^^^^^^^^^^^^^^^

                  ction de la comparaison des natures physiques.
```

f_ Construction de bAD.
^^^^^^^^^^^^^^^^^^^^^^^

Analyse par cas sur la comparaison hierarchique des potentialites.

2_3. Algorithme.
----

cfr point 2_3 pour l'essentiel.

2_4. Passage module logique --> module physique.

Ce passage est nettement plus clair que le precedent et ne suscite que le commentaire suivant :

si une operation est de niveau hierarchique plus eleve qu'une autre,elle se verra attribuer un identifiant entier plus petit;il en va de meme pour les natures physiques des entites --> d'ou inversion : 'plus eleve' devient '<',...

---

2. Démarche de conception des modules de niveau 1.

2.1. Le module de gestion du graphe dynamique (dg)

COMMENTAIRES SUR DG.C

Le module dg a pour mission de gerer la structure representant le graphe dynamique.

Il offre ainsi aux niveaux superieurs des fonctions de manipulation de cette structure et des informations qu'elle contiant.L'ensemble des fonctions ainsi offert est bien sur extensible a souhait.

On a pu mettre en evidence onze fonctions de manipulation d'une telle structure (onze sous-modules).

1_ Sous-module dg_create.

Le sous-module dg_create a pour mission de creer la structure representant le graphe dynamique.

1_1. Specifications.

Arg. : ----

Pre. : ----

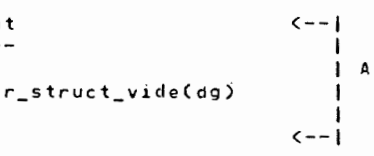
Result.: dg : la structure representant le graphe dynamique.

Post. : dg est vide (aucune action dynamique n'y est presente).

1_2. Indications sur la conception de l'algorithme A.

Tres simple : une seule pseudo_instruction qui est crear_struct_vide(dg)

1_3. Algorithme.



1_4. Passage module logique --> module physique.

En vue d'une reprise eventuelle,la structure sera scindee en deux :

- une structure C dg_f en memoire centrale,

dg_f = taille de tab (nombre d'actions présentes)
tableau tab [L_A] d'actions dynamiques
avec L_A definie a 100.

creer_struct_vide devient des lors :

```
dg_f.taille <-- 0
creer_fichier_vide (dg_tab).
```

2_ Sous-module dg_copy.

Le sous-module dg_copy a pour mission,lors d'une reprise,de creer la structure dg et de lui adjoindre les informations qui y etaient presentes avant la desactivation du logiciel.

2_1. Specifications.

Arg. : dg_info : l'ensemble des informations presentes dans dg avant desactivation.

Pre. : le logiciel a ete active au moins une fois.

Result.: dg

Post. : dg comprend dg_info.

2_2. Indications sur la conception de l'algorithme B.

Tres simple : deux pseudo-instructions (cfr 2_3.)

2_3. Algorithme.

```
Debut                          <--|
-----                             |
                                  |
creer_struct_vide (dg)            | é
remplir_dg_avec (dg_info)         |
                                  |
Fin.                           <--|
----
```

2_4. Passage module logique --> module physique.

dg_info aura ete mise dans dg_tab avant desactivation,si bien que le remplissage de dg_f se fait a partir de dg_tab (cfr 1_4.concernant dg_f et dg_tab).

On a vu au point 1_4. que dg_f etait une structure comprenant :

- le tableau dont les elements ont pour mission de memoriser une action dynamique.

Representation d'une action dynamique :

- on a deja opte pour une structure representant un ensemble de predicats (cfr. COMPAR.COMMENT); on choisit des entiers pour representer les identifiants des entites agissantes,entites subissantes,operations dynamiques,entites parametres,operations statiques.

- il y a deux ensembles de predicats : le premier t_p[0][L_P] contient n_p[0] predicats et porte sur l'operation dynamique,le second t_p[1][L_P] contient n_p[1] predicats et porte sur l'operation statique.

3_ Sous-module dg_get_p.

Le sous-module dg_get_p a pour mission de fournir l'ensemble de predicats portant sur l'operation dynamique d'une action dynamique donnee.

3_1. Specifications.

Arg. : pos : position de l'action dynamique dans dg.
^^^^^^^^
dg : graphe dynamique.

Pre. : pos est une position valide et referencie bien une action de dg.
^^^^^^^^

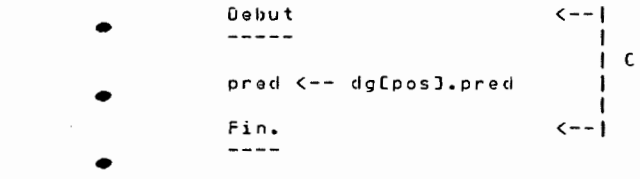
Result.: pred : ensemble de predicats.
^^^^^^^^

Post. : l'ensemble pred de predicats est l'ensemble de predicats portant
^^^^^^^^ sur l'operation dynamique de la posieme action de dg.

3_2. Indications sur la conception de l'algorithme C.

Tres simple : une seule pseudo-instruction.

3_3. Algorithme.



3_4. Passage module logique --> module physique.

Structure de dg : cfr points 1_4. et 2_4.

4_ Sous-module dg_find_1.

Le sous-module dg_find_1 a pour mission de fournir les positions dans dg des actions dynamiques dont les entites agissante et subissanta sont donnees.

4_1. Specifications.

Arg. : ent1 : entite agissante
^^^^^^^^

ent2 : entite subissante

dg : graphe dynamique.

Pre. : ----
^^^^^^^^

Result.: liste_positions
^^^^^^^^

Post. : - chaque elemant de liste_positions donne la position d'une
^^^^^^^^ action dynamique dont les antites agissante et subissante
sont respectivement ent1 et ent2.

- chaque elemant de liste_positions est unique dans cette liste.

- il n'existe pas d'action dynamique dans dg dont l'entite agissante est ent1 et dont l'antite subissante est ent2 et dont la position n'est pas reprise dans liste_positions.

4_2. Indications sur la conception de l'algorithme D.

Raisonnement par induction sur le nombre d'actions contenues dans dg :

pour chaque action de dg,verifier si elle possede les criteres pour que sa position soit mise dans liste_positions.

4_3. Algorithme.

```
Debut                                                  <--|
-----                                                     |
                                                          |
pour toute action dg[i] de dg                             |
   |                                                      | D
   | si     entite agissante de dg[i] = ent1 et           |
   |        entite subissante de dg[i] = ent2             |
   | alors  ajouter 'i' dans liste_positions              |
   |                                                      |
                                                          |
Fin.                                                   <--|
----
```

4_4. Passage module logique --> module physique.

Optimisation : reduire la taille maximale de liste_positions,d'ou risque
^^^^^^^^^^^^^^ eventuel de ne pas lister toutes les positions possibles,
d'ou introduction de deux informations supplementaires :

1- en entree : position dans dg a partir de laquelle
on commence la recherche (start).

2- en sortie : indicateur d'existence de positions
supplementaires non reprises dans
liste_positions (more).

Il s'ensuit un appauvrissement de la fonctionnalite de depart et les modules superieurs utilisant dg_find_1 devront compenser ce manque par appels successifs de dg_find_1 en jouant sur start et ce jusqu'a ce que more prenne la valeur 'non'.

Les remarques des points 1_4. et 2_4. sont de mise ici aussi (structure dg_f).

Passage des parametres : par pointeurs.

5_ Sous-module dg_get.

Le sous-module dg_get a pour mission de fournir l'action dynamique particuliere se trouvant dans dg dans une position donnee.

5_1. Specifications.

Arg. : pos : position de l'action dynamique desiree.
^^^^^^^^
dg : graphe dynamique.

Pre. : pos est valide et referencie donc bien une action de dg.
^^^^^^^^

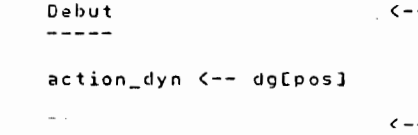
Result.: action_dyn : une action dynamique.
^^^^^^^^

Post. : action_dyn est l'action dynamique referenciee par pos dans dg.
^^^^^^^^

5_2. Indications sur la conception de l'algorithme E.

Tres simple : une seule pseudo-instruction.

5_3. Algorithme.



5_4. Passage module logique --> module physique.

Structure dg_f : cfr points 1_4. et 2_4.

Passage des parametres : par pointeurs.

6_ Sous-module dg_insert.

Le sous_module dg_insert a pour mission d'inserer une nouvelle action dans dg.

6_1. Specifications.

Arg. : action_dyn : une action dynamique.
dg : graphe dynamique.

Pre. : action_dyn n'appartient pas a dg.

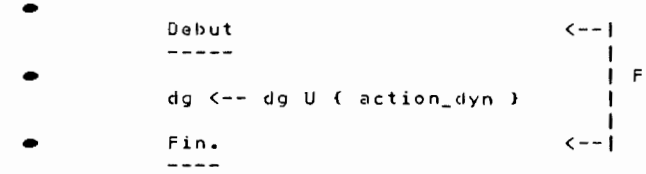
Result.: nouveau dg.

Post. : nouveau dg = dg + action_dyn.

6_2. Indications sur la conception de l'algorithme F.

Tres simple : une seule pseudo-instruction.

6_3. Algorithme.



6_4. Passage module logique --> module physique.

Structure dg_f : cfr points 1_4. et 2_4.

Etant donne que l'on a mis au point 3_4. une limitation sur le nombre d'actions que dg_f peut contenir,le module physique dg_insert contiendra un test sur le nombre d'actions presentes avant insertion et,au cas ou il n'y aurait plus de place pour inserer la nouvelle action,produira un resultat d'exception (CODERR = 10);il s'ensuit une pre_condition supplementaire : CODERR doit etre egal a 0 avant appel de dg_insert.

7_ Sous-module dg_comp.

Le sous_module dg_comp a pour mission de fournir les positions dans dg des actions dynamiques ayant une relation hierarchique donnee avec une action dynamique donnee.

7_1. Specifications.

Arg. : type_relation : indique le type de la relation hierarchique.
act_dyn : une action dynamique.
dg : le graphe dynamique.

Pre. : type_relation prend ses valeurs dans l'ensemble ('<' --> hierarchiquement inferieur,'>' --> hierarchiquement superieur,'=' --> hierarchiquement egal,'<=' --> hierarchiquement inferieur ou egal,'>=' --> hierarchiquement superieur ou egal).

Result.: liste_positions : une liste de positions.

Post. : - chaque element de liste_positions donne la position d'une action dynamique telle que act_dyn entretient la relation "type_relation" avec elle.

- chaque element de liste_positions est unique dans cette liste.

- il n'existe pas d'action dynamique dans dg non reprise dans liste_positions et telle que act_dyn entretient la relation "type_relation" avec elle.

7_2. Indications sur la conception de l'algoritnme G.

a_ Raisonnement par induction : verifier si act_dyn entretient type_ relation avec chaque action dynamique de dg et,le cas echeant,ajouter la position de cette action dynamique dans dg dans liste_positions.

```
pour chaque action dynamique act de dg
   |
   | si     act_dyn <type_relation> a[i]   (GA)
   | alors  ajouter 'i' dans liste_positions
   |
```

b_ Construction de GA.

Pour construire GA,on va se servi d'un sous-module dg_compact qui aura pour tache de comparer deux actions dyn d'un point de vue hierarchique,cad que dg_compact va determiner la rel archique simple ('<','>','=', '<>') existant entre les deux acti iques.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Arg. : act1,act2 : deux actions dynamiques.
^^^^^^^^

Pre. : ----
^^^^^^^^

Result.: res : code_resultat
^^^^^^^^

Post. : res vaut
^^^^^^^^

'<' si act1 est hierarchiquement inferieur a act2.
'>' si act1 est hierarchiquement superieur a act2.
'=' si act1 est de meme niveau hierarachique que act2.
'<>' si act1 est hierarchiquement incomparable a act2.

Suivant le resultat donne par dg_compact,on determinera le resultat de dg_comp.

7_3. Algorithme.

Debut
---

pour chaque action dynamique a[i] de dg
|
| si ((res(dg_compact[act_dyn,a[i]]) = type_relation et
| type_relation appartient a ('<','>','=')) ou
|
| ((res(dg_compact[act_dyn,a[i]]) = '<' ou res(dg_compact[act_dyn,
| a[i]]) = '=') et type_relation = '<=') ou
|
| ((res(dg_compact[act_dyn,a[i]]) = '>' ou res(dg_compact[act_dyn,
| a[i]]) = '=') et type_relation = '>='))
|
| alors ajouter 'i' dans liste_positions.
|

7_4. Passage module logique --> module physique.
---

a_ Codage des resultats de dg_compact
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

'<' --> 1
'>' --> 0
'=' --> 2
'<>' --> 3

b_ Codage de type_relation
^^^^^^^^^^^^^^^^^^^^^^^^^

'<' --> 1
'>' --> 0
'=' --> 2
'>=' --> 3

c_ Structure de dg : cfr points 1_4. et 2_4.
^^^^^^^^^^^^^^^^^^^^^^

d_ Liste_positions : cfr points 4_4. (d'ou start,more).
^^^^^^^^^^^^^^^^^^^^

7_5. Indications sur la construction de dg_compact (H).

Utilisation du module comp_lp de niveau 0 (cfr COMPAR.COMMENT).

Rappel : etant donnes deux ensembles de predicats Ep1 et Ep2, comp_lp determine la relation de contraignance existant entre ces deux ensembles et retourne

'<' si Ep1 est moins contraignant que Ep2
'>' si Ep1 est plus contraignant que Ep2
'=' si Ep1 est aussi contraignant que Ep2
'<>' si Ep1 et Ep2 sont de contraintes incomparables.

Determination immediate de l'incomparabilite :

si les deux actions ne portent pas sur les memes entites agissantes, subissantes et parametres
alors res <-- '<>'
sinon comparer_plus_avant (HA)

Conception de HA.
^^^^^^^^^^^^^^^^

Analyse par cas sur les resultats des comparaisons hierarchiques des operations dynamiques et statiques et de la comparaison de contraignance des deux ensembles de predicats.

7_6. Passage module logique --> module physique.

Structure pour dg et act_dyn : cfr points 1_4. et 2_4.

Codage des resultats : cfr point 7_4.

Hierarchie d'operations : plus une operation est de niveau hierarchique eleve,plus son identifiant entier est petit.

8_ Sous-module dg_delete.

Le sous-module dg_delete a pour mission le retrait d'une action dynamique du graphe dynamique dg.

8_1. Specifications.

Arg. : pos : la position dans dg d'une action dynamique.
^^^^^^^
dg : le graphe dynamique.

Pre. : pos est une position valide au sens ou elle referencie effectivement une action de dg.

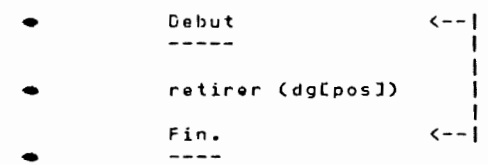
Result.: nouveau dg

Post. : nouveau dg = dg \ { action referenciee par pos }

8_2. Indications sur la conception de l'algorithme I.

Tres simple : une seule pseudo-instruction.

8_3. Algorithme.



8_4. Passage module logique --> module physique.

Technique de retrait :

recopie de chaque action ayant position p > pos de p --> p-1 et decrementation de la taille du dg de une unite.

Structure pour dg : cfr points 1_4. et 2_4.

9_ Sous-module dg_putf.

Le sous-module dg_putf a pour mission de preparer la reprise en memorisant sur support stable (cad. conservant les informations meme lorsque le logiciel n'est pas actif) les informations contenues dans dg.

9_1. Specifications.

Arg. : dg : le graphe dynamique

Pre. : ----

Result.: dg_info

Post. : dg_info est sur support stable.

Raisonnement par induction :

pour chaque action a[i] de dg
|
| ajouter a[i] dans dg_info
|

9_3. Algorithme.
---

cfr point 9_2.

9_4. Passage module logique --> module physique.

Structure pour dg : cfr points 1_4. et 2_4.

10_ Sous-module dg_auto.

Le sous-module dg_auto a pour mission de donner les positions dans dg de toutes les actions susceptibles de fournir une raison d'autoriser un acces dynamique donne (suscaptible d'autoriser --> hierarchiquement superieur ou egal en supposant un ensemble de predicats ep vide).

10_1. Specifications.

Arg. : acc_d : acces dynamique.
^^^^^^^^
dg : graphe dynamique.

Pre. : ----
^^^^^^^^

Result.: liste_positions : une liste de positions.
^^^^^^^^

Post. : - chaque element de liste_positions indique la position dans dg d'une action susceptible de fournir l'autorisation a l'acces dynamique donne.
^^^^^^^^

- chaque element de liste_positions est unique dans cette liste.

- il n'existe pas d'action de dg susceptible de fournir l'autorisation a l'acces dynamique donne dont la position ne soit pas reprise dans liste_positions.

10_2. Indications sur la conception de l'algorithme K.
---

a_ Raisonnement par induction sur les elements de dg

chaque action a[i] de dg

```
                    | alors  ajouter 'i' dans liste_positions.
                    |
```

b_ Conception de KA.

verification des conditions a la susceptibilite d'autoriser :

memes entite agissante,subissante,parametre +

operat_dyn de a[i] hierarcniquement superieure ou egale
a operat_dyn de acc_dyn +

operat_stat de a[i] hierarchiquement superieure ou egale
a operat_stat de acc_dyn.

10_3. Algorithme K.

Debut

```
pour chaque action dynamique a[i] de dg
    |
    | si     ( memes ent_ag,ent_sub,ent_param et
    |          operat_dyn(a[i]) hierarch.sup.ou egal operat_dyn(acc_dyn) et
    |          operat_stat(a[i]) hierarch.inf.ou egal operat_stat(acc_dyn))
    |
    | alors  ajouter 'i' dans liste_positions.
    |
```

Fin.

10_4. Passage module logique --> module physique.

Structure pour dg : cfr points 1_4. et 2_4.

liste_positions : cfr point 4_4.

Hierarchie d'operations : plus une operation est de niveau hierarchique
eleve,plus son identifiant entier est petit.

11_ Sous-module dg_delelem.

Le sous-module dg_delelem a pour mission de retirer du graphe dynamique dg toutes les actions dont l'entite agissante,l'entite subissante ou l'entite parametre correspond a une entite donnee.

11_1. Specifications.

Arg. : ent : identifiant d'une entite.
^^^^^^^

dg : graphe dynamique.

^^^^^^^^

Result.: nouveau dg
^^^^^^^^

Post. : nouveau dg = dg \ (action dont ent_ag,ent_sub ou ent_param
^^^^^^^^ correspond a ent).

11_2. Indications sur la conception de l'algorithme L.

Raisonnement par induction sur les elements de dg :

pour chaque action a[i] de dg
|
| si ent_ag ou ent_sub ou ent_param de a[i] egale ent
|
| alors retirer a[i] de dg.
|

11_3. Algorithme.
---

cfr point 11_2.

11_4. Passage module logique --> module physique.

Structure pour dg : cfr points 1_4. et 2_4.

Technique de retrait : cfr point 8_4.

2.2. Le module de gestion du graphe statique (sg)

COMMENTAIRES SUR SG.C

Le module sg a pour mission de gerer la structure representant le graphe statique.
Il offre aux niveaux superieurs des fonctions de manipulation de cette structure et des informations qu'elle contient.
L'ensemble des fonctions ainsi offert est extensible a souhait.

On a pu mettre en evidence douze fonctions de manipulation d'une telle structure (douze sous-modules).
Ces sous-modules s'apparentent pour la plupart aux sous-modules de dg,si bien que,pour la plupart d'entre eux,nous renverrons le lecteur au fichier DG.COMMENT (travail semblable sur une autre structure).

1_ Sous-module sg_create.

cfr dg_create

2_ Sous-module sg_copy.

cfr dg_copy

3_ Sous_module sg_exist.

Le sous_module sg_exist a pour mission de determiner s'il existe dans sg une action statique hierarchiquement superieure ou egale a une action statique donnee.

3_1. Specifications.

Arg. : act_stat : une action statique.
^^^^^^^^
sg : le graphe statique.

Pre. : ----
^^^^^^^^

Result.: exist : indicateur d'existence.
^^^^^^^^

Post. : - exist = 'oui' s'il existe dans sg une action hierarchiquement
^^^^^^^^ superieure ou egale a act_stat.

- exist = 'non' s'il n'existe pas dans sg d'action hierarchiquement superieure ou egale a act_stat.

3_2. Indications sur la conception de l'algorithme A.

3_3. Algorithme.

```
Debut                                                    <--|
-----                                                       |
                                                            |
liste_positions <-- sg_comp(act_stat,'>=')                  | A
si     liste_positions est vide                             |
alors  exist <-- 'non'                                      |
sinon  exist <-- 'oui'                                      |
                                                            |
Fin.                                                     <--|
----
```

3_4. Passage module logique --> module physique.

Structure pour action statique :
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

meme choix que pour action dynamique mis a part qu'ici il n'y a plus d'operation dynamique ni d'entite parametre et que le seul ensemble de predicats porte sur l'operation statique.

Liste_positions :
^^^^^^^^^^^^^^^^

meme remarque que pour le point 4_ de dg_comp,d'ou appel de sg_comp avec parametre supplementaire indiquant que l'on commence la recherche au debut du graphe statique.(start = 0).

4_ Sous-module sg_get_p.
---

cfr dg_get_p

5_ Sous-module sg_find_1.
---

cfr dg_find_1

6_ Sous_module sg_get.
---

cfr dg_get

7_ Sous-module sg_insert.
---

cfr dg_insert

8_ Sous-module sg_comp (+ sg_compact).

(+ sg_compact)

9_ Sous-module sg_delete.
---

cfr dg_delete

10_ Sous-module sg_putf.
---

cfr dg_putf

11_ Sous-module sg_auto.
---

cfr dg_auto

12_ Sous-module sg_delelem.
---

cfr dg_delelem

2.3. Le module de gestion de la table d'entités (ent)

| COMMENTAIRES SUR ENT.C |

Le module ent a pour mission de gerer la table d'entites du systeme.

Il offre aux niveaux superieurs des fonctions de manipulation de cette table et des informations qu'elle contient.

On a pu mettre en evidence sept fonctions de manipulation de cette table (sept sous-modules).

Un element de cette table se compose de l'identifiant et de la nature physique d'une entite.

1_ Sous-module ent_create.

Le sous-module ent_create a pour mission de creer la table d'entites.

1_1. Specifications.

Arg. : ----

Pre. : ----

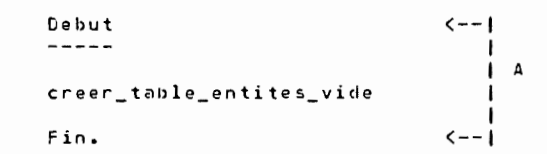
Result.: ent_table : table d'entites.

Post. : ent_table est vide (ne contient aucune entite).

1_2. Indications sur la conception de l'algorithme A.

Une seule psaudo-instruction : creer_table_entites_vide.

1_3. Algorithme.



1_4. Passage module logique --> module physique.

En vue d'une reprise (desactivation puis reactivation) de ce logiciel,deux structures seront utilisees pour representer la table d'entites :

- un fichier ent_tab en memoire secondaire.

On memorisera le contenu de ent_f dans ent_tab avant desactivation et,lors d'une reactivation,on chargera ent_f a partir de ent_tab (ent_f est la structure de travail).

Structure C representant la table d'entites :
^^

entier : taille du tableau tab

tableau tab[L_TAENT] de --> identifiant nominal : char[80]
--> identifiant entier : entier
--> nature physique : entier

avec L_TAENT declare a 50.

2_ Sous-module ent_copy.

Le sous-module ent_copy a pour mission,lors d'une reprise,de creer la table d'entites et de lui adjoindre les informations qui y etaient presentes avant la desactivation du logiciel.

2_1. Specifications.

Arg. : ent_info : ensemble des informations presentes dans la table avant desactivation.

Pre. : le logiciel a ete active au moins une fois (il s'agit bien d'une reactivation).

Result.: ent_table.

Post. : ent_table contient ent_info.

2_2. Indications sur la conception de l'algorithme B.

Deux pseudo-instructions correspondant

1_ a la creation d'une table vide.

2_ au remplissage de la table avec ent_info.

2_3. Algorithme.

2_4. Passage module logique --> module physique.

Le remplissage de la table ent_f se fera a partir du fichier ent_tab
(cfr point 1_4.).

3_ Sous-module ent_get_c.

Le sous-module ent_get_c a pour mission de fournir l'identifiant entier et le type d'une entite dont on donne le nom,au cas ou cette entite existe.

3_1. Specifications.

Arg. : ent : identifiant nominal d'une entite.
^^^^^^^^

ent_table : table d'entites.

Pre. : ----
^^^^^^^^

Result.: trouve : indicateur d'existence d'entite.
^^^^^^^^

id_ent : identifiant entier d'une entite.

type : identifiant d'une nature physique.

Post. : - trouve = 'oui' s'il existe une entite E de la table d'entites
^^^^^^^^ dont l'identifiant nominal correspond a ent,trouve = 'non'
dans le cas contraire.

- si trouve = 'oui',id_ent et type sont respectivement l'identifiant entier et l'identifiant de la nature physique de cette entite.

3_2. Indications sur la conception de l'algorithme C.

Raisonnement par induction sur les elements de la table :

pour chaque element E de ent_table
|
| si ent de E = ent
| alors |
| | trouve <-- 'oui'
| | id_ent <-- id_ent de E
| | type <-- type de E
| |

Initialisation : trouve <-- 'non'

3_3. Algorithme.

3_4. Passage module logique --> module physique.

Structure pour ent_table : cfr point 1_4.

Comparaison caractere par caractere de ent et ent de E.

4_ Sous_module ent_get_i.

Le sous-module ent_get_i a pour mission de fournir l'identifiant nominal et la nature physique d'une entite dont on donne l'identifiant entier.

Specifications.

Arg. : id_ent : identifiant entier d'une entite.
ent_table : table d'entites.

Pre. : ----

Result.: trouve : indicateur d'existence d'entite.
ent : identifiant nominal d'une entite.
type : identifiant de nature physique.

Post. : - trouve = 'oui' s'il existe une entite E de la table d'entites dont l'identifiant entier correspond a id_ent,trouve = 'non' dans le cas contraire.

- si trouve = 'oui', ent et type sont respectivement l'identifiant nominal et l'identifiant de la nature physique de cette entite E.

Pour le reste, il existe une similitude etroite avec ent_get_c,si bien que nous renvoyons le lecteur au point 3_.

5_ Sous-module ent_insert.

Le sous-module ent_insert a pour mission d'inserer un nouvel element dans la table d'entites.

5_1. Specifications.

Arg. : elem : nouvel elemant.
ent_table : table d'entites.

^^^^^^^^

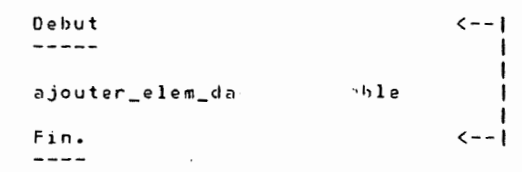
Result.: nouvelle ent_table.
^^^^^^^^

Post. : nouvelle ent_table = ent_table U (elem)
^^^^^^^^

5_2. Indications sur la conception de l'algorithme D.

Une seule pseudo-instruction : ajouter_elem_dans_ent_table.

5_3. [illegible].



5_4. Passage module le --> module physique.

Structure pour ent_ta [illegible]fr point 1_4.

Etant donne que la structure physique ent_f est limitee en taille (L_TAENT), le module physique ent_insert contiendra un test sur le nombre d'elements presents avant insertion et,au cas ou il n'y aurait plus de place pour inserer le nouvel element,produira un resultat d'exception (CODERR = 15);il s'ensuit une pre-condition supplementaire : CODERR doit valoir 0 avant appel de ent_ insert.

6_ Sous-module ent_delete.

Le sous-module ent_delete a pour mission de retirer de ent_table une entite donnee.

6_1. Spacifications.

Arg. : id_ent : identifiant d'une entite.
^^^^^^^^

Pre. : id_ent appartient a ent_table.
^^^^^^^^

Result.: nouvelle ent_table.
^^^^^^^^

Post. : nouvelle ent_table = ent_table \ (elem tel que id_ent(elem) =
^^^^^^^^ id_ent)

Raisonnement par induction sur les elements de ent_table :

```
pour chaque element E de ent_table  (b1)
    |
    | si      id_ent(E) = id_ent
    | alors   |
    |         | retirer_E_de_ent_table
    |         | sortir (b1)
    |         |
```

6_3. Algorithme.

cfr point 6_2.

6_4. Passage module logique --> module physique.

Structure pour ent_table : cfr point 1_4.

Choix de l'identifiant : utilisation ici de l'identifiant entier.

Implementation physique de "sortir (b1)" :

variable auxiliaire initialisee a 'non',mise a 'oui' des que l'on a id_ent(E) = id_ent ; bouclage tant que trouve = 'non'.

7_ Sous-module ent_putf.

Le sous-module ent_putf a pour mission de preparer la reprise en memorisant sur support stable (cad. conservant les informations meme lorsque le logiciel n'est pas actif) les informations contenues dans ent_table.

7_1. Specifications.

Arg. : ent_table : la table d'entites.

Pre. : ----

Result.: ent_info.

Post. : ent_info est sur support stable.

7_2. Indications sur la conception de l'algorithme F.

Raisonnement par induction sur les elements de ent_table :

pour chaque element E de ent_table

7_3. Algorithme.

cfr point 7_2.

7_4. Passage module logique --> module physique.

Structure pour ant_table : cfr point 1_4.

L'ajout de l'identifiant nominal dans ent_info se fait caractere par caractere

2.4. Le module de gestion de la table d'états d'entités (tee)

COMMENTAIRES SUR TEE.C

Le module tee a pour mission de gerer la table d'etats d'entite.

Il offre aux niveaux superieurs des fonctions de manipulation de cette table et de ses elements.

Un element de cette table se compose de deux identifiants : celui de l'entite possesseur et celui de l'entite possedee.

On a pu mettre en evidence neuf fonctions de manipulation de cette table (neuf sous-modules).

1_ Sous-module tee_create.

cfr ent_create (pour module logique).

Note sur le passage au module physique.

Choix de structure pour tee_table :

structure C tee_f composee de

--> taille : taille courante du tableau tab

--> tableau tab[L_TATEE] de

--> poss : identifiant de l'entite possesseur.
--> possede : identifiant de l'entite possedee.

avec L_TATEE definie a 100

+ fichier tee_tab pour preparer reprise (memorisation sur support stable).

2_ Sous-module tee_copy.

cfr ent_copy.

3_ Sous_module tee_getl.

Le sous-module tee_getl a pour mission de fournir la liste des entites possedees par une entite donnee ainsi que la liste des positions dans tee_table des elements exprimant la possession de ces entites par l'entite donnee.

3_1. Specifications.

Arg. : tee_table : table d'etats d'entites.

Pre. : ----
^^^^^^^^

Result.: liste_positions : liste de positions.
^^^^^^^^
liste_possedes : liste d'identifiant d'entites possedees.

Post. : - chaque element e[j] de liste_positions correspond a la position
^^^^^^^^ dans tee_table d'un element e[i] dont l'entite possesseur est poss et chaque element e[k] de liste_possedes correspond a l'entite possedee de l'element e[j] de tee_table ayant une position donnee par e[k] de liste_positions.

- chaque element de liste_positions est unique dans cette liste et chaque element de liste_possedes est unique dans cette liste.

- il n'existe pas d'element de tee_table dont l'entite possesseur est poss et dont la position dans tee_table n'est pas reprise dans liste_positions et dont l'entite possedee n'est pas reprise dans liste_possedes.

3_2. Indications sur la conception de l'algorithme A.

Raisonnement par induction sur les elements de tee_table :

```
pour chaque element e[i] de tee_table
   |
   | si     poss de e[i] = poss
   | alors  |
   |        | ajouter '1' dans liste_positions
   |        | ajouter possede de e[i] dans liste_possedes
   |        |
```

3_3. Algorithme.
---

cfr point 3_2.

3_4. Passage module logique --> module physique.

Structure pour tee_table : cfr point 1_.

La remarque du point 4_4 de dg.comment (dg_find_1) concernant liste_positions s'applique ici a liste_positions et liste_possedes. (d'ou mora et start).

Liste_positions et liste_possedes sont reunies dans une meme structure : t_poss.

4_ Sous-module tee_get2.

- - - de fournir la liste des entites

tee_table des elements exprimant la possession de l'entite donnee par ces entites.

Ce sous-module est donc fort semblable au precedent,si bien qu'a ce stade, nous renvoyons le lecteur au point 3_.

5_ Sous-module tee_exist.

Le sous-module tee_exist a pour mission de determiner s'il existe dans tee_table un element exprimant la possession d'une entite possedee donnee par une entite possesseur donnee,et,le cas echeant,de fournir la position de cet alement

5_1. Specifications.

Arg. : tee_table : table d'etats d'entites.

poss : identifiant d'une entite.

possede : identifiant d'une entite.

Pre. : ----

Result.: exist : indicateur d'existence.

pos : position.

Post. : - exist = 'oui' s'il existe dans tee_table un element exprimant la possession de possede par poss et pos est alors la position dans tee_table de cet element.

- exist = 'non' s'il n'existe pas dans tee_table d'element exprimant la possession de possede par poss et,dans ce cas,pos n'a aucune signification.

5_2. Indications sur la conception de l'algorithme 8.

Raisonnement par induction sur les elements de tee_table :

```
pour chaque element e[i] de tee_table  (b1)
    |
    | si     poss de e[i] = poss et possede de e[i] = possede
    | alors  |
    |        | exist <-- 'oui'
    |        | pos   <-- i
    |        | sortir (b1)
    |        |
```

Initialisation : exist = 'non'.

5_3. Algorithme.

5_4. Passage module logique --> module physique.

Structure pour tee_table : cfr point 1_.

Implementation de "sortir (b1)" : bouclage tant que exist = 'non' et que
pas_traite_tous_les_elements_de_tee_table.

6_ Sous-module tee_insert.

cfr ent_insert.

7_ Sous-module tee_delete.

cfr dg_delete.

8_ Sous_module tee_putf.

cfr ent_putf.

9_ Sous-module tee_delelem.

cfr dg_delelem.

2.5. Le module de gestion de la table de hiérarchisation des opérations (tho)

COMMENTAIRES SUR THO.C

Le module tho a pour mission de gerer la table de hierarchisation des operations

Il offre aux niveaux superieurs des fonctions de manipulation de cette table et des informations qu'elle contient.

Un element de cette table se compose de l'identifiant nominal d'une operation et de l'identifiant hierarchique de l'operation.

On a pu mettre en evidence cinq fonctions de manipulation de cette table (cinq sous-modules).

1_ Sous-module tho_create.

cfr ent_create (pour module logique).

Note concernant le passage au module physique.

Structure pour tho_table :

structure C tho_t composee de

--> taille : taille courante du tableau tab

--> tableau tab [L_TATHO] de

--> droit : ch.car. identifiant nominalement l'operation.

--> place : entier identifiant l'operation par son niveau hierarchique.

avec L_TATHO definie a 10.

Pas de fichier ici car le contenu de tho n'evolue pas (init).

2_ Sous-module tho_get_c.

cfr ent_get_c.

3_ Sous-module tho_get_i.

cfr ent_get_i.

4_ Sous-module tho_insert.

5_ Sous-module tho_delete.
---

cfr ent_delete (retrait d'apres l'identifiant entier place).

2.6. Le module de gestion de la table de classes de sécurité (tcs)

COMMENTAIRES SUR TCS.C

Le module tcs a pour mission de gerer la table de classes de securite.

Il offre aux niveaux superieurs des fonctions de manipulation de cette table et de ses elements.

Un element de cette table se compose de l'identifiant d'une entite,de l'identifiant de l'operation representant la potentialite active de cette entite,de l'identifiant de l'operation representant la potentialite passive de cette entite.

On a pu mettre en evidence six fonctions de manipulation de cette table (six sous-modules).

D'un point de vue logique,les sous-modules de tcs sont semblables a ceux de ent (agissent juste sur des tables differentes),si bien que,pour la plupart d'entre eux,nous renvoyons le lecteur au fichier ent.comment.

1_ Sous-module tcs_create.

Le sous-module tcs_create a pour mission de creer une table de classes de securite vide.

Passage au module physique : contrairement a la table d'entites,la table de classes de securite ne reprend qu'un seul identifiant d'entite (identifiant entier).

2_ Sous-module tcs_copy.

cfr ent_copy.

3_ Sous-module tcs_get.

Le sous-module tcs_get a pour mission de fournir les potentialites active et passive ainsi que la position dans la table d'un element correspondant a une entite donnee.

3_1. Specifications.

Arg. : tcs_tabla : la table de classes de securite.
^^^^^^^^

ent : identifiant d'une entite.

Pre. : ent appartient a tcs_table.
^^^^^^^^

Result.: pot_act,pot_pass,pos.

```
  Post.  :  ent de tcs_table[pos] = ent,pot_act de tcs_table[pos] = pot_act,
  ^^^^^^^^  pot_pass de tcs_table[pos] = pot_pass.
```

3_2. Indications sur la conception de l'algorithme A.

Raisonnement par induction sur les elements de tcs_table :

```
pour chaque element e[i] de tcs_table (b1)
    |
    | si     ent de e[i] = ent
    | alors  |
    |        | pot_act  <-- pot_act de e[i]
    |        | pot_pass <-- pot_pass de e[i]
    |        | pos      <-- i
    |        | sortir (b1)
    |        |
```

3_3. Algorithme.
---

cfr point 3_2.

3_4. Passage module logique --> module physique.

Structure pour tcs_table : cfr point 1_ (introduction).

Implementation physique de "sortir (b1)" :

variable auxiliaire trouve initialisee a 'non',mise a 'oui' des que ent de e[i] = ent;bouclage tant que trouve = 'non'.

4_ Sous-module tcs_insert.

cfr ent_insert.

5_ Sous-module tcs_delete.

cfr dg_delete.

6_ Sous-module tcs_putf.

cfr ent_putf.

2.7. Le module de gestion de la table de cohérence du système (coh)

COMMENTAIRES SUR COH.C

Le module coh a pour mission de gerer la table de coherance du systeme.

Il offre aux niveaux superieurs des fonctions de manipulation de cette table et des informations qu'elle contient.

Un element de cette table comprend un type agissant ainsi qu'un certain nombre de types subissant associes chacun a une liste d'indicateurs de coherence physique (un indicateur par operation dynamique et statique differents,hormis 'delegate' et 'abrogate').

On a pu mettre en evidence trois fonctions de manipulation de cette table (trois sous_module).

1_ Sous-module coh_create.

cfr ent_create (pour le module logique).

Note sur le passage au module physique.

Structure pour coh_table :

Structure C coh_f composee de

--> taille : taille courante du tableau tab

--> tableau tab[L_TACOH] de

--> type_ag : identifiant d'une nature physique agissante.

--> lng : longueur du tableau L

--> tableau L[3] de

--> type_sub : identifiant d'une nature physique subissante.

--> tableau t[7] de ('oui' ouy 'non')

avec L_TACOH definie a 7.

Pas de fichier ici car le contenu de coh_f est fixe et reinitialise par init a chaque activation.

2_ Sous-module coh_accord.

Le sous-module coh_accord a pour mission de determiner la coherence physique de l'execution d'une operation par une entite de nature physique donnee sur une entite de nature physique donnee.

Arg. : coh_table : la table de coherence physique.
^^^^^^^^

droit : identifiant d'une operation.

type_ag : identifiant de la nature physique de l'entite agissante.

type_sub : identifiant de la nature physique de l'entite subissante.

Pre. : droit different de 'delegate' et droit different de 'abrogate'.
^^^^^^^^

Result.: accord : indicateur de coherence.
^^^^^^^^

Post. : accord = coh_table[type_ag,type_sub,droit].
^^^^^^^^

2_2. Indications sur la conception de l'algorithme A.

Une seule pseudo-instruction :

accord <-- coh_table[type_ag,type_sub,droit].

2_3. Algorithme.

cfr point 2_2.

2_4. Passage module logique --> module physique.

Structure pour coh_table : cfr point 1_. (adaptation de l'algorithme sur cette structure).

Il se peut que type_ag ou type_sub n'appartienne pas a la structure --> si l'on ne trouve pas (variable trouve), accord <-- 'non' (fait a l'initialisation).

3_ Sous_module coh_insert.

cfr ent_insert.

2.8. Le module de gestion de la table de listes d'utilisations (act)

COMMENTAIRES SUR ACT.C

Le module act a pour mission de gerer la table d'utilisation du systeme.

Il offre ainsi aux niveaux superieurs des fonctions de manipulation de cette table et de ses elements.

Un element de cette table se compose de'une liste d'entites.

On a pu mettre en evidence sept fonctions de manipulation de cette table (sept sous-modules).

1_ Sous-module act_create.

cfr ent_create (pour module logique).

Note concernant le passage au module physique.

Structure pour act_table :

1_ fichier act_tab memorisant les informations de maniere stable en vue d'une reprise.

2_ structure C act_f composee de

--> taille : taille courante du tableau tab

--> tableau tab [L_TAACT] de

--> lng : longueur de la liste_tableau

--> tableau ent[L_U] d'entiers identifiant les entites.

avec L_U definie a 10 et L_TAACT definie a 20.

2_ Sous-module act_copy.

cfr ent_copy.

3_ Sous-module act_get.

cfr ent_get_i (l'identifiant est ici celui de la premiere entite de la liste d'utilisations).

n.b.: passage au module physique : la structure utilisee pour l'argument identifiant est aussi celle du resultat.

4_ Sous-module act_insert.

cfr ent_insert.

5_ Sous-module act_delete.

cfr ent_delete (l'identifiant est la premiere entite de la liste d'utilisations).

6_ Sous-module act_putf.

cfr ent_putf.

7_ Sous-module act_delelem.

Le sous-module act_delelem a pour mission de reduire les listes d'utilisations jusqu'a et y compris l'endroit ou se trouve l'identifiant d'une entite donnee.

7_1. Specifications.

Arg. : act_table : table de listes d'utilisations.
^^^^^^^^
ent : identifiant d'entite.

Pre. : ----
^^^^^^^^

Result.: nouvelle act_table.
^^^^^^^^

Post. : nouvelle act_table = act_table ou les listes d'utilisations ont
^^^^^^^^ ete tronquees a partir de la position de l'identifiant ent dans
le cas ou celles-ci contenaient cet identifiant.

7_2. Indications sur la conception de l'algorithme A.

Raisonnement par induction sur les elements de act_table :

pour chaque liste d'utilisations l[i] de act_table (b0)
|
| tronquer_si_necessaire (AA)
|

Conception de AA.

Raisonnement par induction sur les elements d'une liste d'utilisations :

pour chaque entite e[k] de l[i] (b1)
|
| si e[k] = ent

```
        |         | sortir (b1)
        |         |
```

7_3. Algorithme.
---

cfr point 7_2.

7_4. Passage module logique --> module physique.

Structure pour act_table : cfr point 1_.

Troncature d'une liste d'utilisations :

un tableau C commencant avec l'indice 0,lorsque l'on rencontre ent a l'indice k dans le tableau correspondant a une liste d'entites,on met la longueur ing de ce tableau a k.

Implementation physique de "sortir (b1)" :

on utilise une variable supplementaire effectue initialisee a 'non' entre b1 et b0,mise a 'oui' si e[k] = ent et un bouclage sur les elements d'une liste d'utilisations l[i] tant que effectue vaut 'non'.

2.9. Le module de gestion de l'identification à attribuer à la prochaine entité que l'on voudra créer (num)

COMMENTAIRES SUR NUM.C

Le module num a pour mission de gerer le numero identifiant que l'on attribuera a une nouvelle entite.

Il offre aux niveaux superieurs des fonctions de manipulation de ce numero identifiant.

On a pu mettre en evidence trois fonctions de manipulation de ce numero (trois sous-modules).

1_ Sous-module num_create.

cfr ent_create (pour le module logique).

Note concernant le passage au module physique.

num_tab est un fichier servant a memoriser le numero avant desactivation du logiciel,afin de garder ce meme numero pour une reactivation ulterieure.

2_ Sous-module num_copy.

cfr ent_copy.

3_ Sous-module num_putf.

cfr ent_putf.

3. Démarche de conception des modules de niveau 2.

3.1. Le module garantissant la cohérence (gar-coh)

COMMENTAIRES SUR GAR_COH.C

Le module gar_coh a pour mission de determiner si l'insertion d'une action dynamique ou statique viole ou ne viole les regles de coherence.

On a pu mettre deux fonctions en evidence (deux sous-modules).

1_ Sous-module verif_dyn.

Le sous-module verif_dyn a pour mission de determiner si l'insertion d'une action dynamique viole ou ne viole pas les regles de conerence.

1_1. Specifications.

Arg. : act_dyn : une action dynamique (3 entites + droit_d)

Pre. : CODERR = 0

Result.: coherent,CODERR

Post. : - coherent = 'oui' si l'insertion de act_dyn ne viole pas les regles de coherence physique;coherent = 'non' dans le cas contraire.

- CODERR = 0 si coherent = 'oui'

- CODERR = 23 + ensemble 1 infos.erreur si au moins deux des entites agissante,subissante,parametre sont identiques.

- CODERR =24 + ensemble 2 infos.erreur si incoherence physique.

n.b. : - ensemble 1 infos.erreur

--> ENT_ERR : identifiant entier entite agissante.
--> OBJ_ERR : identifiant entier entite subissante.
--> OR_ERR : identifiant entier operation.
--> PAR_ERR : identifiant antier entite parametre.

- ensemble 2 infos.erreur

--> ENT_ERR : identifiant entier entite agissante.
--> OBJ_ERR : identifiant entier entite subissante.
--> OR_ERR : identifiant entier operation.

1_2. Indications sur la conception de l'algorithme A.

Analyse par cas sur l'identite d'entite :

[illegible] dyn) = ent sub(act_dyn) ou ent_ag(act_dyn) = ent_param

```
   alors  |
          | CODERR = 23
          | ensemble 1 infos.erreur <-- elements de act_dyn
          |
   sinon    verif_phys  (AA)
```

Conception de AA.

Decomposition et sequencement :

1_ determiner les natures physiques de ent_ag et ent_sub.

2_ determiner la coherence physique de (type(ent_ag),type(ent_sub),droit (act_dyn).

D'ou verif_phys devient :

```
t = ent_get_i (ent_ag,ent,type_ag)
t = ent_get_i (ent_sub,ent,type_sub)
coherent <-- coh_accord (type_ag,type_sub,droit de act_dyn)
si     coherent = 'non'
alors  |
       | CODERR <-- 24
       | ensemble 2 infos.erreur <-- elements de act_dyn.
       |
```

1_3. Algorithme.
---

cfr point 1_2.

1_4. Passage module logique --> module physique.

Isolement de AA dans une fonction C separee car on s'en servira encore pour verif_stat.

2_ Sous-module verif_stat.

Le sous-module verif_stat a pour mission de determiner si l'insertion d'une action statique viole ou pas les reglas de coherence physiques et relatives aux classes de securite.

2_1. Specifications.

Arg. : act_stat : entite agissante,entite subissante et droit d'une action statique.

Pre. : CODERR = 0.

Result.: CODERR : indicateur d'erreur.

coherent : indicateur de coherence.

Post. : - si l'insertion projetee ne viole pas les regles de coherence et
^^^^^^^^ si ent_ag est differente de ent_sub
alors CODERR = 0 et coherent = 'oui'.

- si les regles de coherence physiques sont violees
alors CODERR = 24 + remplissage ensemble 1 infos.erreur.

- si les regles relatives aux classes de securite sont violees
alors CODERR = 21 (violation pot.active) ou CODERR = 22 (violation pot.passive) et remplissage ensemble 2 infos.erreur.

- si ent_ag = ent_sub
alors CODERR = 25 et remplissage ensemble 1 infos.erreur.

n.b. : - ensemble 1 infos. erreur

--> ENT_ERR : identifiant entier entite agissante.
--> OBJ_ERR : identifiant entier entite subissante.
--> OR_ERR : identifiant entier operation.

- ensemble 2 infos.erreur

--> ENT_ERR : identifiant entier de l'entite pour laquelle la potentialite active ou passive est violee.
--> OR_ERR : identifiant de l'operation violant cette potentialite.

2_2. Indications sur la conception de l'algorithme B.

Analyse par cas sur l'identite des deux entites :

si ent_ag(act_stat) = ent_sub(act_stat)
alors |
| CODERR = 25
| ensemble 1 infos. erreur <-- elements de act_stat
|
sinon pousser_verif_plus_avant (BA)

Conception de BA.

Analyse par cas sur le resultat de la verification des regles relatives aux classes de securite :

si regles_classes_securite_verifiees (BAA)
alors verifier_regles_physiques (BAB)

Conception de BAA.

Decomposition et sequencement :

1_ determiner les potentialites de ent_ag

2_ verifier les regles relatives aux classes de securite de ent_ag.

3_ si verifie alors

3_2_ verifier les regles relatives aux classes
de securite de ent_sub.

D'ou l'algorithme BAA suivant :

```
tcs_get (ent_ag(act_stat),pot,pos)
si      pot.active < droit(act_stat)
alors   |
        | CODERR = 22
        | ensemble 2 infos.erreur <-- (ent_ag(act_stat),droit(act_stat))
        |

sinon   |
        | tcs_get (ent_sub(act_stat),pot,pos)
        | si      pot.passive < droit(act_stat)
        | alors   |
        |         | CODERR <-- 22
        |         | ensemble 2 infos.erreur <-- (ent_sub(act_stat),
        |         |                                      droit(act_stat))
        |         |
```

Conception de BAB.
^^^^^^^^^^^^^^^^^^

identique a AA du point 1_2.

2_3. Algorithme.
----

cfr point 2_2.

2_4. Passage module logique --> module physique.

Algorithme BAB devient un appel a verif_phys (point 1_4).

Comparaison entre potentialite et droit :

etant donne que plus un droit est de niveau hierarchique eleve,plus son identifiant hierarchique est petit,il faut inverser les signes de comparaison pour passer au module physique : '<' devient '>'.

Algorithme BAA : egalement mis dans une fonction C separee.

3.2. Le module autorisateur (autorisateur)

COMMENTAIRES SUR AUTORISATEUR.C

Le module autorisateur du niveau 2 de ce logiciel a pour mission de determiner si un acces donne est autorise ou non.
Il recoit en entree une action act et l'identifiant de l'entite intelligente ent_intell qui est a l'origine de cet acces (pas necessairement celle qui perpetue l'acces).

Definition de act.

act comprend les elements suivants :

- entl : identifiant de l'entite agissante de l'acces.
- ent2 : identifiant de l'entita subissante da l'acces.
- droit_d : identifiant de l'operation principale de l'acces (dynamique ou statique).
- param : identifiant de l'entite parametra de l'acces (cas dynamique).
- droit_s : identifiant de l'operation secondaire de l'acces (operation statique,cas acces dynamique).
- n_pred : longueur du tableau de predicats portant sur l'operation dynamique (cas acces dynamique).
- t_p : tableau de predicats (cfr COMPAR.COMMENT).

1_ Specifications.

Arg. :
- act : acces
- dg : graphe dynamique.
- sg : graphe statique.
- tee_table : table d'etats d'entites.
- act_table : table des listes d'utilisations.
- ent_intell : entite intelligente a l'origine de l'acces.

Pre. : CODERR = 0

Result.:
- autorise : indicateur d'autorisation.
- CODERR : indicateur d'erreur.

Post. :
- autorise = 'oui' et CODERR = 0 si l'acces act est autorise (selon les lois d'autorisation).
- autorise = 'non' et CODERR = 43 si l'acces act n'est pas autorise (selon les lois d'autorisation).

2_ Indications sur la conception de l'algorithme A.

1_ determiner si chacune des entites de la liste d'utilisations identifiee par ent_intell peuvent utiliser (use) toutes les autres entites qui la suivent dans cette liste.

2_ si 1_ est verifie,determiner si chacune des entites de la liste d'utilisations identifiea par ent_intell sont autorisees a faire l'acces act.

3_ si 1_ ou 2_ n'est pas verifie,mettre CODERR a 43.

Algorithme A.
^^^^^^^^^^^^^

```
Debut
-----

autorise <-- aut_util(ent_intell)  (AA)
si     autorise = 'oui'
alors  |
       | autorise <-- aut_finale(act,ent_intell)  (AB)
       | si     autorise = 'non'
       | alors  CODERR <-- 43
       |
sinon  CODERR <-- 43

Fin.
----
```

Conception de AA.
^^^^^^^^^^^^^^^^^

Decomposition et sequencement :

1_ obtenir la liste d'utilisations identifiee par ent_intell.

2_ proceder aux verifications.

```
Debut
-----

obtenir_liste_util l[i] (ent_intell)   (AAA)
determiner_autorisat  (AAB)

Fin.
----
```

Conception de AAA.
^^^^^^^^^^^^^^^^^^

utilisation de act_get.

Conception de AAB.
^^^^^^^^^^^^^^^^^^

Raisonnement par induction sur les elements de l[i] :

pour chaque entite e[j] de l[i] et tant qu'autorise = 'oui'

Initialisation : autorise <-- 'oui'

Conception de AABA.
^^^^^^^^^^^^^^^^^^^

Raisonnement par induction sur une sous-liste d'utilisations commencant en e[j+1] :

pour chaque entite e[k] de la sous-liste de l[i] commencant en e[j+1]
|
| autorise <-- stat(e[k],ent2 de act,use)
|

Nous arretons ici l'exposition detaillee de la conception et informons le lecteur que 'stat' determine l'autorisation en utilisant les modules de niveau 1 et en faisant appel a verif_pred qui,via verif_p,verifie si les predicats d'une action susceptible d'autoriser cet acces sont verifies.

Conception de AB.
^^^^^^^^^^^^^^^^^

Analyse par cas suivant le type d'acces (type de l'operation principale de act [droit_d]);le cas ou droit_d est statique se negocie d'un seul tenant (sauf cas du create).

Pour chaque cas on raisonne par induction sur les elements de la lista d'utilisations identifiee par ent_intell :

Debut
-----

l[i] <-- act_get(ent_intell)
si droit_d de act = delegate
alors |
| pour chaque entite e[k] de l[i] (b1)
| |
| | autorise <-- delegat (act,e[k]) (ABA)
| | si autorise = 'non'
| | alors sortir (b1)
| |

si droit_d de act = abrogate
alors |
| pour chaque entite e[k] de l[i] (b2)
| |
| | autorise <-- abrogat (act,e[k]) (ABB)
| | si autorise = 'non'
| | alors sortir (b2)
| |

si droit_d de act = grant
alors |
| pour chaque entite e[k] de l[i] (b3)
| |
| | autorise <-- gran (act,e[k]) (ABC)
| | autorise = 'non'

```
        |     | si     autorise = 'non'
        |     | alors  sortir  (b3)
        |     |
```

```
si      drcit_d de act = revoke
alors   |
        | pour chaque entite e[k] de l[i]   (b4)
        |     |
        |     | autorise <-- revok (act,e[k])     (ABD)
        |     | si     autorise = 'non'
        |     | alors  sortir  (b4)
        |     |

si      droit_d de act est statique
alors   |
        | si     droit_d de act est different de create
        | alors  |
        |        | pour chaque entite e[k] de l[i]  (b5)
        |        |     |
        |        |     | autorise <-- stat (act,e[k])  (ABE)
        |        |     | si     autorise = 'non'
        |        |     | alors  sortir  (b5)
        |        |     |
        | sinon  autorise <-- 'oui' (tout le monde peut creer)
        |
```

Conceptions de ABA et ABB (identiques).
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Analyse par cas en fonction de la possession de param de act par ent1 de act ,de la possession de param de act par ent2 de act,de la possession de ent2 de act par ent1 de act.

Utilisation de tee_exist du niveau 1 :

```
      si     exist <-- tee_exist (ent1 de act,param de act) = 'oui'
      alors  |
             | si     exist <-- tee_exist (ent2 de act,param de act) = 'oui'
             | alors  |
             |        | si    exist <-- tee_exist (ent1 de act,ent2 de act) =
             |        |                                                 'oui'
             |        | alors autorise <-- 'oui'
             |        | sinon autorise <-- 'non'
             |        |
             | sinon  autorise <-- 'oui'
             |
      sinon  autorise <-- 'non'.
```

Conception de ABC.
^^^^^^^^^^^^^^^^^^

Par rapport a ABA,s'ajoute ici les faits qu'une entite peut gratifier a une entite qu'elle possede toutes les actions statiques qu'elle detient (utilisation de sg_exist) et qu'il n'est pas necessaire de posseder l'entite parametre pour gratifier pourvu que cette gratification soit validee par le graphe dynamique (utilisation des fonctions de dg) et que les predicats soient verifies (verif_pred).

Conception de ABD.
^^^^^^^^^^^^^^^^^^

... qu'il n'est pas necessaire de posseder

le graphe dynamique (utilisation des fonctions de dg) et que les predicats soient verifies (verif_pred).

Conception de AAE.
^^^^^^^^^^^^^^^^^^

cfr conception de AABA.

3_ Algorithme.
---

cfr point 2_.

4_ Passage module logique --> module physique.

Choix de structures pour :

act_table : cfr ACT.COMMENT

tee_table : cfr TEE.COMMENT

sg : cfr SG.COMMENT

dg : cfr DG.COMMENT

Choix de structure pour act (parametre de autorisateur) :

--> ent1 : entier
--> ent2 : entier
--> droit_d : entier
--> param : entier
--> droit_s : entier
--> n_pred : entier
--> t_p[L_P]: tableau de predicats (cfr COMPAR.COMMENT)

Mapping du module logique vers les modules physiques :

A --> autorisat, AA --> aut_util, AB --> aut_finale, ABA --> delegat,

ABB --> abrogat, ABC --> gran, ABD --> revok, ABE --> stat.

Definition de structures de passage entre modules physiques :

cfr code source --> d'ou remplissage de ces structures avant appel.

Passage de la plupart des parametres :

utilisation de pointeurs.

---

4. Démarche de conception des modules de niveau 3 (execut)

```
-------------------------------------------------------------------------------
|                    COMMENTAIRES SUR EXECUT.C                                |
-------------------------------------------------------------------------------

Le module execut du niveau 3 de ce logiciel a pour mission d'executer les acces
demandes s'ils sont autorises ou de produire un code d'erreur s'ils sont refuses

Il recoit en entree une action act,l'identifiant de l'entite intelligente ent_
intell qui est a l'origine de cette action et eventuellement l'identifiant d'un
type d'entite (cas du create : type de la nouvelle entite).

   Definition de act.
   ------------------

   act comprend les elements suivants :

       entl : identifiant de l'entite agissante.

       ent2 : identifiant de l'entite subissante.

       droit_d : identifiant du droit principal (statique ou dynamique).

       param : identifiant de l'entite parametre.(cas dynamique)

       droit_s : identifiant du droit statique secondaire(cas dynamique)

       n_p[2] : longueurs des ensembles de predicats dynamiques (n_p[1]),
                statiques (n_p[2]).

       t_p[2] : ensembles de predicats statiques (t_p[2]) et dynamiques(t_p[1])
                ( cfr COMPAR.COMMENT ).

  1_ Specifications.
  ------------------

    Arg.    : act         : action
    ^^^^^^^^^
              ent_intell  : identifiant d'une entite intelligente.

              type        : identifiant de la nature physique d'une nouvelle
                            entite.

              sg          : graphe statique.

              dg          : graphe dynamique.

              ent_table   : table d'entites.

              tcs_table   : table de classes de securite.

              act_table   : table de listes d'utilisations.

              tee_table   : table d'etats d'entites.

              coh_table   : table de coherence physique.


    Pre     :  - CODERR = 0.
```

d'utilisations identifiee par ent_intell.

Result.: - effectue : indicateur d'accomplissement de l'action.
^^^^^^^^

- CODERR : indicateur d'erreur.

Post. : - effectue = 'oui' et CODERR = 0 si l'action act est autorisee et
^^^^^^^^ si son execution n'a viole aucune regle de coherence.

- effectue = 'non' et CODERR = 20 si l'action act est un create d'entite de type 'user' et que ent1 est differente de 'superuser'.

- effectue = 'non' et CODERR = 26 si l'action act est un create et que l'une des entites de la liste d'utilisations identifiae par ent_intell n'a pas la potentialite active de creer.

2_ Indications sur la conception de l'algorithme A.

Analyse par cas : si droit_d de act est termine,on ne doit pas verifier si c'est autorise (la terminaison est toujours autorisee) :

```
si     droit_d de act = termine                              <--|
alors  effectue <-- termin (ent_intell)  (AA)                   |
sinon  |                                                        |
       | autorise <-- autorisat (act,ent_intell)               | A
       | si     autorise = 'non'                                |
       | alors  effectue = 'non'                                |
       | sinon  effectuer_act (AB)                              |
       |                                                     <--|
```

Conception de AA.

Utilisation des primitives de act du niveau 1 (retourne toujours 'oui').

Conception de AB.

Analyse par cas sur droit_d de act ; un write ou read donne ici tout de suite une raponse favorable (pas de modifications des elements [tables] du systeme).

```
effectue <-- 'oui'                                              <--|
si     droit_d de act = create                                     |
alors  effectue <-- crea (act,type,ent_intell)  (ABA)              |
si     droit_d de act = delete                                     |
alors  effectue <-- dele (act)                  (ABB)              |
si     droit_d de act = use                                        |
alors  effectue <-- util (act,ent_intell)       (ABC)              | AB
si     droit_d de act = delegate                                   |
alors  effectue <-- delega (act)                (ABD)              |
si     droit_d de act = abrogate                                   |
       effectue <-- abroga (act)                (ABE)              |
                                                                   |
```

```
    si      droit_d de act = revoke                                  |
    alors   effectue <-- revoc (act)                  (ABG)    <--|
```

Conception de ABA.

Il s'agit ici,avant d'effectuer la creation,de verifier les regles de coherence speciales de la creation :

```
    si      type = 'user'
    alors   |
            | si      ent_intell = ent et ent = 'superuser'
            | alors   coherent <-- 'oui'
            | sinon   |
            |         | coherent <-- 'non'
            |         | CODERR <-- 20
            |         |
    sinon   si      toutes les entites de la liste d'utilisations identifiee par
                    ent_intell ont la potentialite active de creer
            alors   |
                    | si      (coherent <-- coh_accord (type de ent1 de act,type,
                    |                                   create)) = 'non'
                    | alors   CODERR = 27 + ensemble infos.erreur
                    | sinon   make_creat (act,type)   (ABAA)
            sinon   CODERR <-- 26 + ensemble infos.erreur
```

Conception de ABAA.

Utilisation de : tee_insert,sg_insert (insertion de possessions et de droits statiques delete pour le createur et tous les possesseurs de ce createur) et tee_get2 du niveau 1.

Conception de ABB.

Effacement de l'entite dans les graphes statique et dynamique,dans ent_table, tcs_table et tee_table grace a l'utilisation des fonctionnalites offertes par le niveau 1.

Conception de ABC.

Ajout de l'entite nouvellement utilisee (ent2 de act) dans la liste d'utilisations identifiee par ent_intell : utilisation de act_get,act_delete et act_insert du niveau 1.

Conception de ABD.

Il s'agit de verifier tout d'abord si l'action statique que l'on veut deleguer respecte les lois de coherence (utilisation de verif_stat) et ensuite de verifier si,pour toutes les entites presentes dans l'option de gratification,les lois de coherence sont respectees (utilisation de verif_dyn).
Si tout cela est respecte,on insere dans le graphe dynamique toutes les actions dynamiques deleguees et l'action statique deleguee s'insere dans [illegible] containes regles (ins_sg,ins_dg).

- s'il existe deja une action hierarchique superieure ou egale a l'
  action que l'on veut inserer,il n'est nul besoin d'aller plus loin.

- s'il existe des actions hierarchiques inferieures a l'action que l'on
  veut inserer,on supprime ces actions avant d'effectuer l'insertion.

(utilisation de sg_comp,dg_comp,sg_insert,dg_insert,sg_delete,dg_delete).

Conception de ABE.

Il n'y a ici aucune coherence d'insertion a verifier car il s'agit d'un retrait; le retrait logique est toujours effectue (meme si le retrait physique ne l'est pas [l'action n'existe pas]).

La regle de retrait est la suivante :

- on retire toutes les actions de niveau hierarchique inferieur ou egal
  a celui de l'action que l'on veut retirer (utilisation de sg_comp,
  dg_comp,sg_delete,dg_delete du niveau 1).

Conception de ABF.
---

cfr ABD (ici,il n'y a qu'une action statique a inserer et aucune action dynamique).

Conception de ABG.
---

cfr ABE (ici,il n'y a qu'une action statique a supprimer et aucune action dynamique).

3_ Algorithme.
---

cfr point 2_.

4_ Passage module logique --> module physique.

Structures pointees pour passage des parametres entre modules physiques.

Ajout a differents niveaux de tests et codes d'erreur en raison de la limitation en place des differents graphes et tables (modules physiques testant,d'un point de vue limitation de place,les insertions et permettant une insertion en tout ou rien correspondant a un seul appel de execut [v_ins_sg,v_ins_dg,v_ovf]).

---

5. Démarche de conception des modules de niveau (exec-int)

COMMENTAIRES SUR EXEC_INT.C

Le module exec_int a pour mission de convertir les identifiants nominaux en identifiants entiers afin de faciliter la manipulation des droits et entites que l'on referencie dans les actions que l'utilisateur entreprend.

1_ Specifications.

Arg. :
- action : action avec identifiants nominaux.
- type : identifiant entier du type d'une nouvelle entite (cas create).
- ent_int : identifiant nominal de l'entite intelligente.
- dr_plus : identifiant nominal de la potentialite active.
- dr_moins : identifiant nominal de la potentialite passive.
- ent_table : table d'entites.
- tcs_table : table de classes de securite.

Pre. :
- CODERR = 0

Result.:
- CODERR + erreur : indicateurs d'erreur.
- act : action avec identifiants entiers.
- nouveau ent_table : table d'entites.
- nouveau tcs_table : table de classes de securite.

Post. :
- CODERR = 0 et erreur = 'non' si toutes les entites et droits referencias existent dans le systeme (sauf cas create),si l'entite que l'on veut creer n'existe pas encore dans le systeme (cas du create),si l'entite qui termine son execution est la derniere entite de la liste d'utilisations identifiee par ent_intell (cas du termina).
- erreur = 'oui' et CODERR = 32 si une antite referenciee n'existe pas dans le systeme.
- erreur = 'oui' et CODERR = 33 si une operation referenciee n'existe pas dans le systeme.
- erreur = 'oui' et CODERR = 38 si l'entite que l'on veut creer porte l'identifiant nominal d'une entite existant deja dans le systeme (create).
- erreur = 'oui' et CODERR = 39 si l'entite qui veut terminer n'est pas la derniere entite de la liste d'utilisations. (termine)

2_ Indications sur la conception de l'algorithme A.

a_ Transformation de ent_intell,ent1 de action,droit_d de action (deux sous-modules de transformation : trsforme et trsformd).

b_ Si pas d'erreur en a_,analyse par cas :

```
si      droit_d de act est dynamique
alors   |
        | transformation de ent2,param,droit_s de action
        | si      toujours pas d'erreur
        | alors   |
        |         | si      droit_d de act = delegate
        |         | alors   trsform.options gratification
        |         | si      toujours pas d'erreur
        |         | alors   |
        |         |         | trsform.predicats statiques
        |         |         | si      toujours pas d'erreur
        |         |         | alors   trsform.predicats dynam.
        |         |         |
sinon   si      droit_d de act different de termine
        alors   |
                | si      droit_d de act = create
                | alors   |
                |         | si      ent2 de action existe deja
                |         | alors   CODERR <-- 38 + ens.infos
                |         |                       erreur
                |         | sinon   trsform.dr_plus,dr_moins et
                |         |         si pas d'erreur,insertion
                |         |                 dans ent et tcs.
                |         |
                | sinon   trsform.ent2 de action
                |
                sinon   si      ent1 de act n'est pas la derniere entite
                                de la liste d'utilisations identifiee
                                par ent_intell
                        alors   CODERR <-- 39 + ens.infos erreur.
```

3_ Algorithme.

cfr point 2_.

4_ Passage module logique --> module physique.

Creation : test et code d'erreur supplementaire se referant a un eventuel overflow dans ent et tcs.

6. Démarche de conception des modules du gestionnaire d'erreurs (gest-err)

COMMENTAIRES SUR GEST_ERR.C

Le module gest_err a pour mission le traitement d'une erreur,cad. que sur base du code d'erreur qu'on lui donne,il determine quels sont les messages a envoyer au terminal.Ce module peut recevoir d'autres informations complementaires d'erreur.

1_1. Specifications.

Arg. : CODERR + ensemble d'infos complement. : caracterise l'erreur.

Pre. : CODERR != 0

Result.: CODERR
messages_ecran

Post. : CODERR = 0 (remis a 0 apres tout traitement d'erreur).
messages_ecran correspondent a l'erreur.

Pour mener cette mission a bien,gest_err disposera d'une table indiquant pour chaque valeur possible de CODERR tout au moins une partie du message d'erreur destine a l'utilisateur,l'autre partie etant contenue dans

ENT_ERR : identifiant entier d'une entite.

OBJ_ERR : identifiant entier d'une subissante composant un acces non valide.

OP_ERR : identifiant hierarchique d'une operation.

PAR_ERR : identifiant entier d'une entite parametre.

TYP_ERR : identifiant d'une nature physique.

ENT_ERRC : identifiant nominal d'une entite.

OBJ_ERRC : identifiant nominal d'une entite subissante.

OP_ERRC : identifiant nominal d'une operation.

PAR_ERRC : identifaiant nominal d'une entite parametre.

Ces informations sont delivrees,le cas echeant,par le module qui constate l'erreur.

1_2. Indications sur la conception de l'algorithme A.

Analyse par cas sur la valeur de CODERR.

1_3. Passage module logique --> module physique.

Les fonctions C de l'interface utilisees pour afficher les messages d'erreur :

imp_l : affichage d'une ligne de 80 caracteres.

imp_d : affichage de 10 caracteres (nom d'une operation ou d'une nature physique).

7. Démarche de conception des modules d'initialisation (init)

COMMENTAIRES SUR INIT.C

Le module init a pour mission d'initialiser le systeme,cad. de creer les tables et de les remplir de facon adequate.
En ce qui concerne ent_table,tcs_table,tee_table,sg,dg,act_table et num_table, init les crea avec les informations contenues sur support stable s'il s'agit d'une reactivation ou,dans le cas ou il ne s'agit pas d'une reactivation,init met la superuser dans ent_table,tcs_table et act_table (autres vides).
En ce qui concerne coh_table et tho_table, elles sont toujours creees et remplies de la meme maniere :

tho_table

contient --> (delegate,100)
--> (abrogate,101)
--> (grant,102)
--> (revoke,103)
--> (create,0)
--> (delete,1)
--> (write,2)
--> (use,3)
--> (read,4)
--> (termine,5)

termine servant a indiquer qu'une entite est desactivee (sert a retirer des listes d'utilisations une entite qui a termine son travail).

coh_table

type_ag = user
lng = 3
--> type_sub = user
t = (oui,oui,non,non,non,oui,oui)

--> type_sub = programme
t = (oui,oui,oui,oui,oui,oui,oui)

--> type_sub = zone_d
t = (oui,oui,oui,non,oui,non,non)

type_ag = programme
lng = 2
--> type_sub = programme
t = (oui,oui,oui,oui,oui,oui,oui)

--> type_sub = zone_d
t = (oui,oui,oui,non,oui,non,non)

type_ag = zone_d
lng = 0

n.b.: les sept elements de t correspondent a une operation donnee
(create,delete,write,use,read,grant,revoke)

Utilisation de :
---

sous_modules de niveau 1.

Passage module physique :
---

ajout de fonctions d'affectation de constantes
string a une variable de type chaine de caractere.

---

8. Démarche de conception des modules de query (query)

COMMENTAIRES SUR QUERY.C

Le module query a pour mission de repondre aux questions suivantes :

1_ quelles sont les entites pouvant agir sur une entite donnee,et de quelle facon ?

2_ quelles sont les entites sur lesquelles une entite donnee peut agir,et de quelle facon ?

3_ quel est le chemin d'action d'une entite sur une autre ?

4_ quel est le predicat statique associe au droit statique d'une action dynamique ?

Pour des raisons d'extension eventuelle a des questions de type "Si je fais telle ou telle action,quelles sont les entites pouvant agir sur une entite donnee,et de quelle facon ?",le module query travaille sur ses propres tables initialisees a partir des tables du systeme d'autorisations proprement dit lors de l'activation du query.

Algorithme.

a_ Initialisation des tables du query a partir de celles du systeme d'autorisations proprement dit (sous-module q_init).

b_ Tant que pas fin

| b1_ Affichage menu et attente reponse (menu_2 de interf).
|
| b2_ Construction reponse en fonction de la demande (1ere question q_sub,2eme question q_ag).
|

Algorithmes de q_ag et q_sub.

Utilise lect_ent et aff_res (interf).

Analyse proprement dite : algorithme recursif. (base sur l'algorithme donne dans l'expose d'ACTEN).

Verification de coherence : suppose les predicats verifies et evite le bouclage dans l'analyse.

9. Démarche de conception des modules de l'interface (interf)

COMMENTAIRES SUR INTERF.C

Le module interf a pour mission d'effectuer les echanges d'informations (saisie de donnees,affichage de messages d'erreur,de resultats) avec l'exterieur

Il est a cette fin constitue de sept fonctions (sept sous-modules).

1_ Sous-module imp_l.

Le sous-module imp_l a pour but d'afficher un message de 80 caracteres.

2_ Sous-module imp_d.

Le sous-module imp_d a pour but d'afficher un message de 10 caracteres (correspond a un droit ou une nature physique).

3_ Sous-module saisie.

Le sous-module saisie a pour but de saisir les donnees correspondant a une action et se subdivise en differentes parties suivant le droit principal de l'action,le type des predicats.

4_ Sous-module menu_1.

Le sous_module menu_1 a pour but d'afficher le menu principal et de lire le choix de l'utilisateur.

5_ Sous-module menu_2.

Le sous-module menu_2 a pour but d'afficher le menu de la partie query du logiciel et de lire le choix de l'utilisateur.

6_ Sous-module lect_ent.

La sous-module lect_ent a pour but de lire au terminal le nom de l'entite concernee par une demande au 'query'.

7_ Sous-module aff_res.

Le sous-module aff_res a pour mission d'afficher la reponse a une demande au 'query'.

Lors du passage a l'architecture physique,ce sous-module se scinde en aff_res ,menu_rep,aff_act,aff_ch,aff_pred,aff_p,conv_np,conv_h.

10. Démarche de conception des modules de coordinateur (acten)

COMMENTAIRES SUR ACTEN.C

Le module acten a pour mission de coordonner les executions des differents modules du logiciel afin d'assurer les fonctionnalites du systeme d'autorisations.

Algorithme A.

```
init
fin <-- 'non'
tant que fin = 'non'
   |
   | affich+lect(menu_princ) : menu_1 (reponse)
   | si     reponse = 'action'
   | alors  action   (AA)
   | si     reponse = 'query'
   | alors  query
   | si     reponse = 'quit'
   | alors  fin <-- 'oui'
   |
```

Algorithme AA.

a_ Saisie

b_ Exec_int

```
c_ Si     CODERR different de 0
   alors  gest_err
   sinon  |
          | execut
          | si     CODERR different de 0
          | alors  gest_err + elimination de ent2 de ent et tcs(cas create)
          |
```

Passage au module physique.

AA devient un module physique a part entiere.

Passage des parametres par pointeurs pour les strucutures.

C H A P I T R E II

GUIDE DE L'UTILISATEUR

Le logiciel en question s'exécute au nom de "acten" et propose le menu principal suivant à l'utilisateur :

1. Exécution d'une opération
2. Query (analyse du système)
3. Quit (sortie du logiciel)

L'utilisateur répondra à l'aide des chiffres 1, 2 ou 3: toute réponse dont le premier caractère non blanc n'est ni 1 ni 2 ni 3 reconduira à ce menu.

1. Orientation "exécution d'une opération"

Le système connaît les opérations suivantes :
delegate, abrogate, grant, revoke, create, delete, write, use, read et termine
où termine permet d'indiquer au système qu'une entité utilisée "use" par une autre veut terminer de s'exécuter. (ce qui conduit à son retrait de la chaîne d'utilisation adéquate).

Après saisie de l'opération désirée, le système saisit également l'entité agissante, l'entité subissante (sauf pour termine), l'entité paramètre (cas d'une opération dynamique), l'opération statique secondaire (cas d'une opération principale dynamique), ainsi que les prédicats éventuels sur la ou les opérations, et la liste d'options de gratification (cas d'une opération delegate ou abrogate).

La liste d'options de gratification est limitée à 10 entités. Pour terminer cette liste, il suffit d'entrer "no" comme nom d'entité.

Les types de prédicats que le système connaît sont :

- temporel
- existence - action - statique
- existence - action - dynamique
- potentialité - active
- potentialité - passive

- possession
- physique
- no (indique qu'il n'y a plus de prédicats)

Le nombre de prédicats portant sur une opération est limité à trois.

Le dernier élément que le système saisit est le nom de l'entité de contrôle de l'action, c'est-à-dire le nom de l'utilisateur qui est à la base (directement ou indirectement (chaîne d'utilisation)) de cette action.

Si, après saisie, le logiciel réaffiche le menu principal sans autre message, celà signifie que l'accès est autorisé par le système; dans le cas contraire, un message d'erreur est délivré à l'utilisateur.

2. Orientation "query (analyse du système)"

La partie "query" de ce logiciel permet à l'utilisateur de prendre connaissance de l'état dans lequel se trouve le système d'autorisations, c'est-à-dire des actions susceptibles d'être autorisées par le système.
A cette fin, un menu indique à l'utilisateur quels sont les moyens d'investigation.

1. Liste d'actions exécutables par une entité.
2. Liste d'actions qu'une entité peut subir.
3. Quit (retour au menu principal).

L'utilisateur entre le chiffre 1, 2 ou 3 suivant son choix.
Dans les cas 1 ou 2, l'utilisateur voit tout d'abord apparître la première action faisant partie de la réponse à sa requête: l'utilisateur entre une chaîne de caractères quelconque au niveau de "vu" : pour indiquer qu'il a pris acte de cette action.
Le système affiche alors un dernier menu permettant de guider l'utilisateur dans la visualisation de la réponse :

1. Action suivante
2. Action précédante
3. Chemin correspondant à l'action courante

4. Prédicats éventuels sur l'action statique courante (cas dynamique)
5. Quit (retour au menu principal du query)

Au cas où il n'y aurait pas d'action suivante (action courante = dernière action de la réponse) ou d'action précédente (action courante = première action de la réponse). le système réaffiche ce menu.

Le chemin correspondant à l'action courante donne la liste des entités intermédiaires utilisées en chaîne pour arriver à cette action.

Les prédicats éventuels sur l'opération statique ne sont présents que dans le cas où l'action est dynamique: en effet, le système suppose, lors de son analyse que les prédicats principaux sont vérifiés (prédicats portant sur l'opération dynamique dans le cas d'une action dynamique ou prédicats portant sur l'opération statique dans le cas d'une action statique).

Après chaque affichage (d'une action, d'un chemin ou d'un ensemble de prédicats), le système attend que l'utilisateur lui indique qu'il a pris acte de l'information (vu :) avant de repasser au menu secondaire du query.

C H A P I T R E III

LISTE DES MESSAGES D'ERREUR

1. - dépassement de la capacité du graphe dynamique
2. - dépassement de la capacité du graphe statique
3. - dépassement de la capacité de la table de hiérarchisation des opérations
4. - dépassement de la capacité de la table d'états d'entités
5. - dépassement de la capacité de la table de classes de sécurité
6. - dépassement de la capacité de la table d'entités
7. - dépassement de la capacité de la table de cohérence
8. - dépassement de la capacité de la table d'utilisations
9. - dépassement de la capacité d'une liste d'utilisations

Ces 9 premiers messages d'erreur ne sont liés qu'à l'implémentation physique du système (limitation spatiale des tables et graphes).

10. - seul le superuser peut créer un utilisateur
11. - surpasse la potentialité active de
12. - surpasse la potentialité passive de

Les messages 11 et 12 peuvent apparaître lorsque l'on essaie d'introduire dans le graphe statique une action dont l'une des entités agissante ou subissante n'a pas la potentialité requise par l'opération statique de cette action.

13. - insertion impossible car pourrait aboutir à un auto-droit

Le treizième message indique qu'entités agissante, subissante et éventuellement (action dynamique) paramètre ne sont pas différentes

14. - insertion physiquement impossible

Ce quatorzième message indique qu'une des entités agissante ou subissante est de nature physique incompatible avec l'action que l'on veut insérer dans un des graphes.

15. - n'a pas la potentialité active de créer
16. - ne peut physiquement pas créer une entité de type

Ce seizième message avertit l'utilisateur que l'entité créatrice n'a pas le droit de créer une entité de nature physique spécifiée lors de la saisie.

17. - n'est pas répertoriée comme entité du système
18. - n'est pas répertoriée comme droit du système
19. - heure non valide

Ce dix-neuvième message indique que l'heure entrée est syntaxiquement ou logiquement incorrecte (syntaxe : cchcc où c = chiffre ou chcc).

20. - est un nom déjà attribué à une entité du système

Le message numéro 20 ne peut intervenir que lors d'une tentative de création d'une nouvelle entité.

21. - ne termine pas une chaîne d'utilisation

Le message numéro 21 indique que l'entité agissante d'une demande d'accès n'est pas au bout d'une chaîne d'utilisation et n'est par conséquent pas active pour le moment.

22. - type erroné

Ce vingt-deuxième message avertit l'utilisateur que la nature physique saisie n'est pas connue du système (le système connaît : "user", "programme", "zone-données").

23. - accès physiquement impossible
24. - accès refusé.